DTIC FILE COPY

(4)

MARINE PHYSICAL LABORATORY

SCRIPPS INSTITUTION OF OCEANOGRAPHY

San Diego, California 92152

NAVIGATION SOFTWARE FOR THE MPL VERTICAL LINE ARRAY

B. J. Sotirin and W. S. Hodgkiss

MPL TECHNICAL MEMORANDUM 409

MPL-U-12/89
March 1989

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MPL TECHNICAL MEMORANDUM 409 [MPL-U-12/89]	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Marine Physical Laboratory 6b. OFFICE SYMBOL (If applicable) MPL	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research Department of the Navy
6c. ADDRESS (City, State, and ZIP Code) University of California, San Diego Scripps Institution of Oceanography San Diego, CA 92152	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable) ONR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-87-K-0225 and N00014-87-C-0127
8c. ADDRESS (City, State, and ZIP Code) Department of the Navy 800 North Quincy Street Arlington, VA 22217-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

NAVIGATION SOFTWARE FOR THE MPL VERTICAL LINE ARRAY

12. PERSONAL AUTHOR(S)

B. J. Sotirin and W. S. Hodgkiss

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
tech memo	FROM ______ TO ______	March 1989	103

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	navigation software, vertical line array, travel time measurements, nonlinear least squares

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the navigation software and demonstrates its operation using data obtained during an experiment in which a large aperture low frequency acoustic array, designed and built at the Marine Physical Laboratory (MPL), was deployed vertically from the Research Platform *FLIP* in the NE Pacific. The array was equipped with a 12 KHz acoustic navigation subsystem. Travel time measurements from near bottom acoustic transponders of known position were received by specific array elements throughout the deployment. These measurements were converted to spatial positions of the array elements by a nonlinear least squares technique. The data collection methods and navigation software programs which locate the transponders, calculate the travel time from detected returns and convert travel times to spatial positions are documented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL W. S. Hodgkiss	22b. TELEPHONE (Include Area Code) (619) 534-1798 22c. OFFICE SYMBOL MPL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

★ U.S. Government Printing Office: 1986—607-044

Navigation Software for the MPL Vertical Line Array

B. J. Sotirin and W. S. Hodgkiss

Marine Physical Laboratory
Scripps Institution of Oceanography
La Jolla, CA 92093

ABSTRACT

This report describes the navigation software and demonstrates its operation using data obtained during an experiment in which a large aperture low frequency acoustic array, designed and built at the Marine Physical Laboratory (MPL), was deployed vertically from the Research Platform *FLIP* in the NE Pacific. The array was equipped with a 12 KHz acoustic navigation subsystem. Travel time measurements from near bottom acoustic transponders of known position were received by specific array elements throughout the deployment. These measurements were converted to spatial positions of the array elements by a nonlinear least squares technique. The data collection methods and navigation software programs which locate the transponders, calculate the travel time from detected returns and convert travel times to spatial positions are documented.

Navigation Software for the MPL Vertical Line Array

B. J. Sotirin and W. S. Hodgkiss

Marine Physical Laboratory
Scripps Institution of Oceanography
La Jolla, CA 92093

Introduction

A high frequency acoustic navigation system is an integral part of the large aperture vertical array deployed in 4700 m of water in the NE Pacific during September 1987. Navigation is defined in this context as the process of locating individual elements of the array in 3-dimension space at any particular time. The method implemented in the array involves a transceiver near the ocean surface which sends unique interrogation signals that are detected by bottom moored transponders who reply with a pulse at 12 KHz. This reply is monitored by the array and the time delay between the initiation and reception of each of the pulses is calculated. Knowing the sound speed in the water column, and the location of the transceiver and the transponders, this travel time defines a slant range between each array receiver and each bottom transponder. A slant range from a known point (a transponder) describes an sphere of possible receiver locations. Intersecting spheres described by the slant ranges from three known points identify a single location if there are no errors. Because there are always sources of error in real data, a least squares filter is implemented to approximate the location by minimizing the squared difference between the calculated and measured values, which defines the error in the assumed position. This report documents the data collection methods and the navigation software used to locate 12 array receivers during the September experiment. The array itself is documented in [Sotirin *et al*, 1988] and [Sotirin and Hildebrand, 1988]. The array navigation system, least squares filter and navigation data analysis are documented in [Sotirin and Hildebrand, 1989].

The navigation processing is separated into three programs. The first program locates the bottom transponders. The second program calculates travel times from a continuous recording of the 12 KHz detected returns for each navigation receiver. The third program uses the travel times and transponder locations output from the first two programs to estimate the spatial positions of the receivers using a non-linear least squares filter. The logic flow and input/output files are detailed for each program in the Appendix referenced in the text followed by a sample run. The software, written in either C or Fortran, is listed in Appendix E.

I. Transponder Localization.

The transponder positions must be surveyed to acquire the location parameters defining the transponder net which are ultimately used to navigate the array. Due to errors in the measured data, an estimation technique (least squares) must be implemented. Due to the nonlinear conversion from travel time space to xyz positional space, the least squares method proceeds iteratively. A data set is obtained containing spatial positions and travel time measurements. If all parameters were known exactly, the travel times

calculated using the spatial positions and the measured value would be identical. This is obviously not the case, and the difference between the calculated and measured values defines the error in the assumed position which is minimized during the iteration. Transponder positions with accuracies of less than a meter are achieved by this method. The description below details the data collection, the program inputs, the least squares implementation, and the resulting transponder positions.

Data Collection. The input data set for the least squares iteration requires initial spatial positions of the transponders and transceiver in meters from an arbitrary origin, and slant ranges between the transponders and transceiver. The data collected in the form of travel times and Global Positioning Satellite (GPS) fixes are transformed into the input parameters required by the least squares filter. GPS fixes are converted into initial xy positions in meters from an arbitrary origin. Travel times are converted to slant ranges with knowledge of the local sound speed profile. The travel times required are normally recorded during an intensive surface ship survey during which the ship criss-crosses the area in which the transponders were deployed recording the travel time data and GPS fixes for its own position. [Spiess, 1985], [Smith *et al*, 1975] A transceiver is either hull mounted or towed on a short line such that its position relative to the ship is known. The transceiver sends a continuous stream of unique transponder interrogation pulses, and the transponder replies are recorded while the ship criss-crosses the area. Using this method, a series of travel times are obtained from a wide variety of ship positions. As the ship crosses over the top of a transponder, an estimate of transponder depth is acquired, and as the transponder baselines are crossed, intertransponder distances are defined.

During the September experiment, although travel times were recorded as described, the 12 KHz receiver was deployed on a 200 m line due to the noise level of the ship. This introduces the interrogator/receiver depth as an unknown defining an underdetermined set of equations which cannot be solved for a unique solution. Since the measurements described above were not sufficient for the transponder survey, the travel time data collected by the navigation equipment installed on *FLIP* were utilized instead. This is unfortunately not an optimum choice of observation configurations [Spiess, 1985] so the results of several simulations devised to assess the effect on the estimated positions are also presented. The horizontal motion of *FLIP* was constrained by a three point moor. The *FLIP* data set provides sufficient range information but the azimuthal component was not well constrained. The vertical component was estimated from the echo sounding depth at *FLIP* and apriori information that the sea floor in the experiment area was relatively flat. The inputs used to locate the transponders are the vertical sound speed profile in the test area, the initial xy positions of *FLIP* from the arbitrary origin, the depth of the interrogation transponder hardwired 'o the *FLIP*, the slant ranges from each transponder to *FLIP* as determined by chart recorder traces and associated errors, the initial transponder xy positions from the arbitrary origin using the satellite fixes and the transponder depths.

The initial GPS positions in latitude and longitude are converted to xy distances in meters from an arbitrary origin. For the September experiment, the origin was chosen as 34° 47' N, 126° 00' W, with x increasing positively toward the east and y increasing positively toward the north. A GPS position of the ship is recorded during each transponder deployment. The transponders have 45 kg negative buoyancy when they are deployed from the fantail of the surface ship to insure that the GPS position of the ship is an accurate initial position of the actual transponder position on the sea floor. Latitude and longitude for each set of travel time measurements (one measurement from each transponder) are also converted to m to provide the initial estimate of the transceiver position during the survey. The 'survey' from *FLIP* had such a limited range that the same initial position was used for all measurement sets. During a normal ship survey, travel time data is collected from a large variety of horizontal positions and individual initial positions are important. If GPS positions are not available, a technique detailed in Appendix D may be used to estimate the initial position from the measured travel time data. Positions in latitude and longitude were converted to meters by calculating the radius of the earth at the position using the following equations [Stacey]:

$$x = r_p\ \delta(lat)\,, \qquad y = r_p\ cos(lat)\ \delta(long)$$

$$r_p = a\ (\ 1 - f\ \ \sin^2(lat)\ )\,, \qquad f = \frac{a-c}{a}$$

where x is the E-W distance in meters of the position from the origin, y is the N-S distance in meters of the position from the origin, r_p is the radius of the earth at the position latitude, $\delta(lat)$ is the difference in latitude in radians between the position and the origin, $\delta(long)$ is similarly the difference in longitude, lat is the latitude of the position, a is the equitorial radius of the earth in meters and c is similarly the polar radius.

The conversion between travel time and xyz positions requires knowledge of the sound speed profile at the experiment site. The local sound speed profile was calculated from measurements of conductivity, temperature and depth. Travel time deviations due to the variations within the thermocline and to refraction of acoustical energy were shown to be negligible for this experiment therefore a constant sound speed (harmonic mean) was used for the conversion. [Sotirin and Hildebrand, 1989]

The travel time measurements used for transponder localization were collected by transmitting the transponder interrogation signals from *FLIP* once an hour for 18 days and recording the returns on a chart recorder. The transceiver is mounted on the bottom (90 m in depth) of *FLIP* and the pulse level is adjusted manually above the ambient noise for consistent transponder replies. Round trip travel times for navigating *FLIP* are measured carefully by hand on the chart recorder output with an estimated random gaussian error of 2-3 ms. The chart recorder trace, set on a one second sweep rate, records the filtered 12 kHz (500 Hz bandwidth) replies received from the transponder being interrogated (Figure 1.1) delayed 6-7 seconds from the interrogation pulse. Each transponder is interrogated individually by transmitting its unique signal, triggered by the chart recorder, once per second for 45 seconds, and notating its reply on the chart by sweep number with color-coded pens (red, green or blue). The figure shows the direct and multipath (surface/bottom bounce) returns for each transponder. Differentiation between the surface and bottom bounces is difficult because the *FLIP* transceiver is the same distance below the surface as the transponder is above the sea floor. The direct return from the red and blue transponders is strong and consistent. Several error modes are evident in the returns from the green transponder however. The sporatic direct return of the green transponder (G_d) is caused by the transponder detecting either the surface or bottom bounce of the transmitted signal. The error was attributed to the transponder because the same transceiver system detects the returns from all transponders and this type of error was not evident on all transponders. Detection of the multipath arrival of the interrogation pulse delays the arrival of the transponder direct return such that it arrives at the same time as the multipath return of a direct interrogation pulse (G_{m1}). The multipath return of the multipath interrogation arrives 120 ms later (G_{m2}) clearly identifying the first error mode. The second error mode of the green transponder is displayed as the smaller perturbations in the time of arrival of the G_d return. It is more difficult to identify because the deviation from the true return is smaller (although still significant) and not as consistent as the first type of error. These error modes were identified during about 20% of the experiment at random intervals, causing concern during array detection post-processing discussed in the next section.

Software Implementation. Once the ingredients for the least squares method have been accumulated, all xyz positions are adjusted until the root mean squared (rms) error satisfies the convergence criteria. This is accomplished in several parts by first maintaining constant transponder positions and perturbing the *FLIP* positions, then holding the current *FLIP* positions constant while perturbing the transponder positions, and finally examining the mean squared error of each *FLIP* position to determine whether it should be preserved as a viable contributor. A general outline of the method is presented in Figure 1.2 and discussed below. This is followed by a detailed description of the conversion from travel times to slant ranges and spatial positions.

The program consists of three concentric stages, the first stage adjusts the xy positions in two inner loops each testing the rms error against specific convergence criteria for the least squares filter, the second

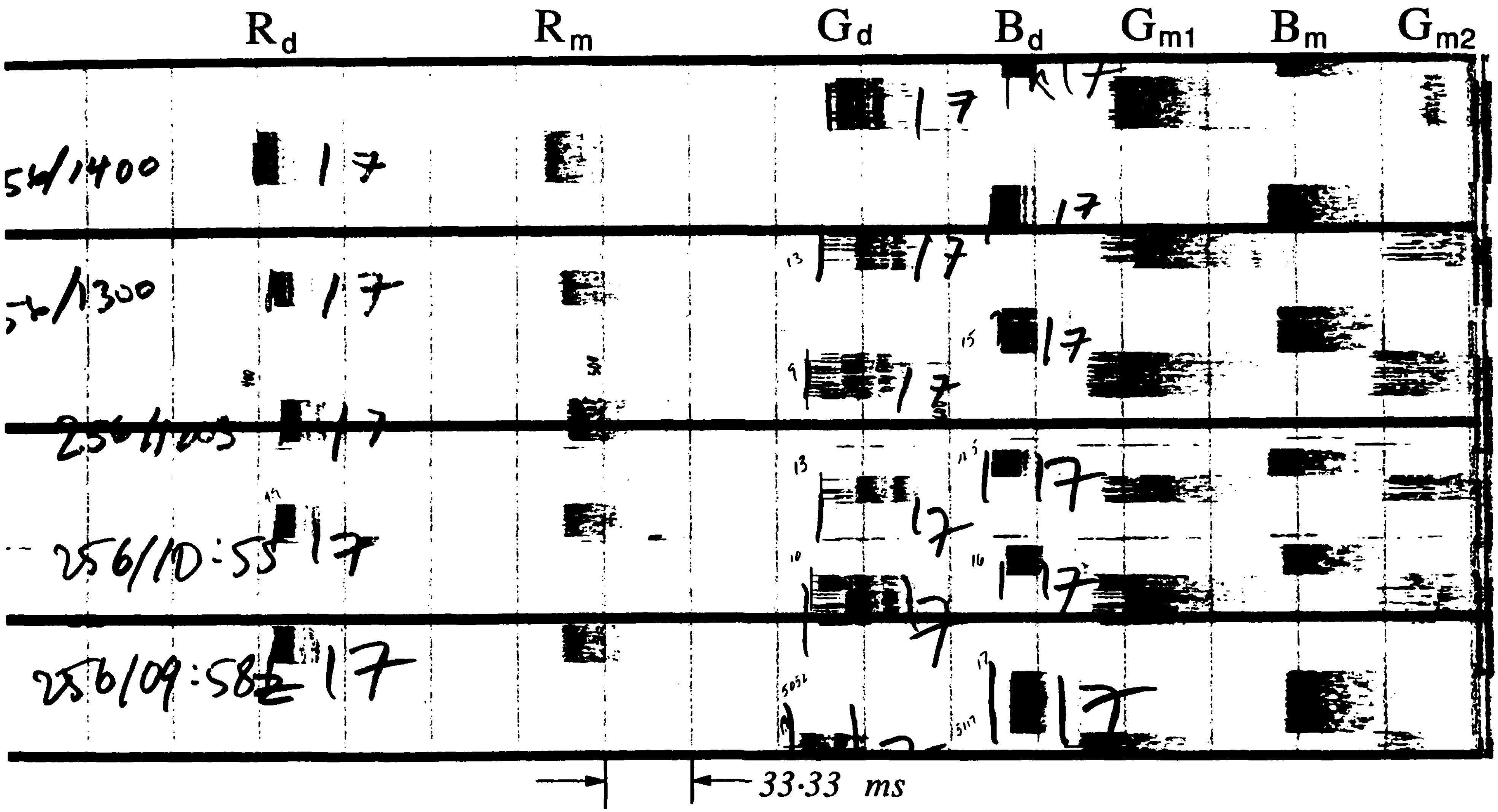

Figure 1.1 ***FLIP* Navigation System Detection of Transponder Replies.** The signals represent the transponder replies plotted by a chart recorder. Each sweep represents one second of round trip travel time, each tic mark is 33.33 ms. Each transponder direct return is marked by Julian day/GMT and by the sweep number with colored pens for easy identification. A normal detection is clear and concise and may be measured by hand to within 2-3 ms. Two types of errors are evident in the returns of the green transponder. The first illustrates a weak direct return G_d, and the spacing between returns G_d, G_{m1} and G_{m2} indicates that the transponder is replying to a bounce of the interrogation signal. The second type of error illustrates smaller scale random returns, inconsistent in occurance and amplitude.

stage tests the total transponder rms error against a second set of convergence criteria, and the third stage tests yet again. The three stages are referred to within the software as stage 1: *xpfil*, stage 2: *xploop* and stage 3: *xpmain*. The two first stage loops shown in Figure 1.2 have similar internal operations. The measured travel times, sound speed profile, *FLIP* depth and transponder depths are considered known, while the xy positions of *FLIP* and the transponders are considered unknown. The measured travel times are converted to slant ranges by multiplying by the harmonic mean of the sound speed profile. This slant range is projected as a horizontal range and compared to the range calculated from the xy positions. For the first loop, the squared difference between these two ranges are summed over the number of transponders for a particular *FLIP* position and the square root is taken to yield the rms error. If the rms error does not satisfy the convergence criteria, then the *FLIP* position is adjusted and the loop repeats with the adjusted position. The adjustment is calculated with the search direction as the negative gradient and the step size as a constant (1.5 m) unless the rms error is less than 1 m at which time the step size begins to decrease; as the minimum is approached, the step size is calculated as a function of the percent change in interated rms error. The convergence criteria for the first stage loops are defined such that an absolute rms error less than 0.15 m, a 0.015 percent change in the iterated rms error, or a maximum number of iterations (30) will terminate the loop and save the current position. These criteria test each adjusted position individually. The first loop is repeated for each *FLIP* position. The second loop of the first stage performs the same cadence maintaining the current *FLIP* positions constant while adjusting the transponder positions. The rms error is calculated over the number of *FLIP* positions, and evaluated using the same criteria. The second loop is repeated for each transponder position. In the second stage, these two loops are initiated again based on the percent reduction (< 0.35%) in the transponder rms error summed over all transponders and total number of iterations (> 30). Upon completion of the first two stages, the rms error for all the transponders is calculated and evaluated according to the following criteria:

1) The rms error * $\sqrt{\text{number of } FLIP \text{ positions}}$ is < 1.0.
2) The percent reduction in rms error is less than a specified value (0.1%).
3) The absolute rms error is less than a specified number (0.75).

If the rms error satisfies any of the above criteria, the current positions are written out and the program is terminated. If the rms error does not satisfy any of the above criteria, then the rms errors associated with each *FLIP* position are examined and any position with an error greater than the transponder rms error times a user specified value (default=2) is deleted from the array and the entire process begins again. The rms error is initialized to 10000 prior to each loop/stage so that unless the absolute error is small, the loop/stage will always be executed more than once.

The conversion from travel time to slant range and the spatial position adjustment is straight forward and executed within the subroutines *xxcor* and *xpfil*. A detailed outline of the program flow is found in Appendix A. The program originates with the Marine Physical Laboratory's DEEPTOW group and has been in existance since the late 1960's. The version documented here is the most recent in a long series and, unfortunately, every programmer has left a mark. Consequently, the subroutines each have unique variable names for the parameters, increasing the confusion in documentation. The text below attempts to maintain the variable names within the subroutines specified. The travel time to slant range conversion is computed in *xxcor*. The measured information is input to the program as a slant range assuming a homogeneous medium with a sound speed of 1500 m/s. This is not normally the case, and for the September test the sound speed profile was measured and input as a data statement in *xxcor* into the array vdp as a horizontally stratified medium. If sound speed corrections are requested by the user, the input slant ranges are converted back to the original time measurement (t) and the slant range contribution for each sound speed layer is summed, returning the corrected slant range (xnew). If the slant range is a depth, for example, the summation starts at the surface and sums the contribution in each layer until the accumulated time (t_a) is equal to the measured travel time. This is an implementation of the following equation in which x is the corrected slant range:

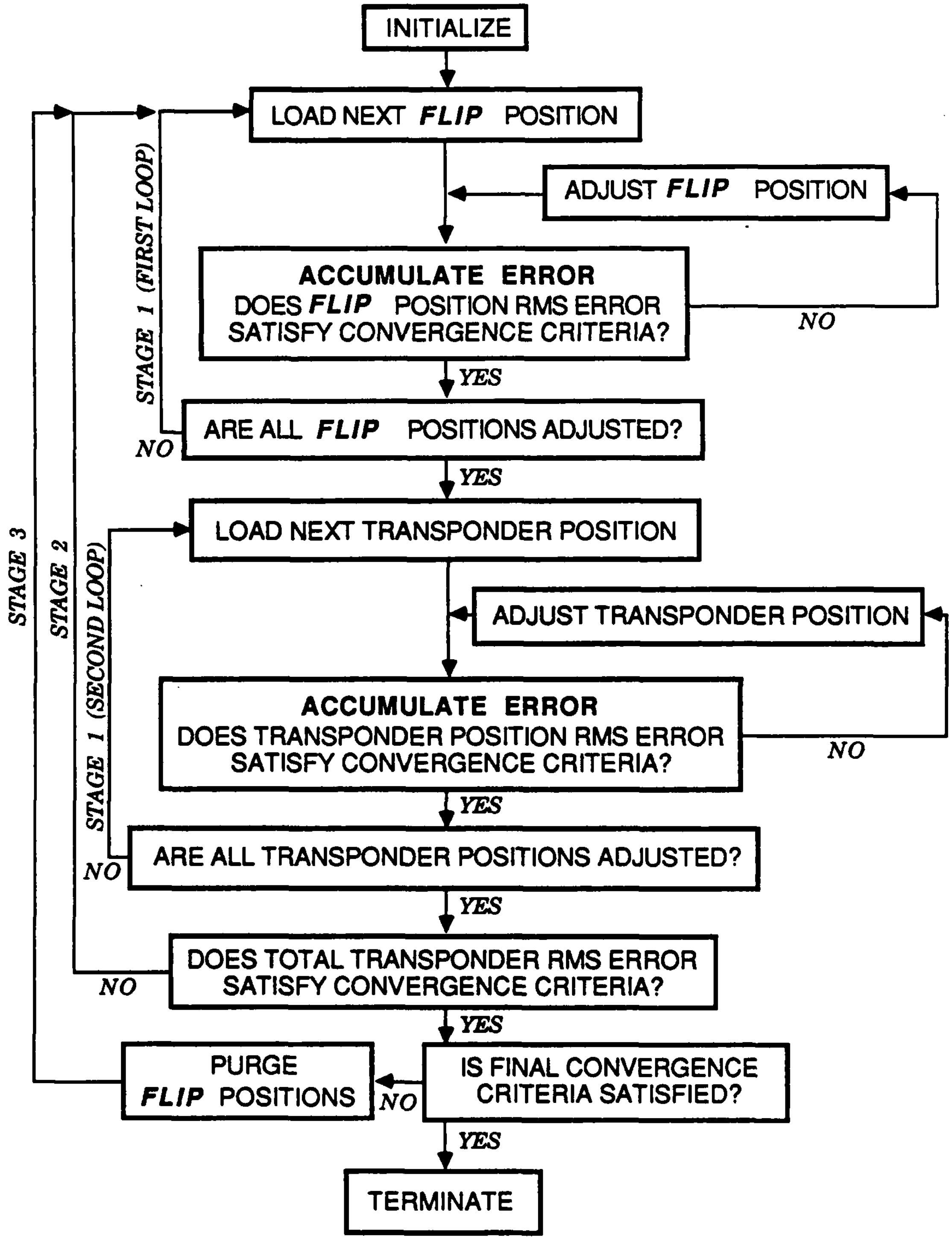

Figure 1.2 **Transponder Localization Program Flow.** A general outline of 3 stages in the least squares program logic is presented. During stage 1, the *FLIP* positions are iterated first, followed by the transponder positions and the rms errors are calculated for the positions iterated and compared to a set of convergence criteria; stage 2 compares transponder rms errors to a second set of convergence criteria; stage 3 compares transponder rms errors against a third set of convergence criteria and determines whether to delete *FLIP* positions with excessive errors.

$$x = c * t = \frac{(z_T - z_0)}{\int_{z_0}^{z_T} \frac{dz}{C(z)}} * t$$

where in terms of the subroutine, z_0 = xdepth = 0, z_T = z is the unknown depth, C(z) = v is the sound speed in the layer, and dz = deltad is the layer depth. When the unknown is a slant range, the implementation is an approximation to the above equation (where dz is the slant range component in the layer assuming the launch angle is constant) which is accurate for the configuration of the data set considered for the September experiment. Errors increase as the horizontal range or projection (defined below) of the slant range increases however, so for larger scale experiments a more accurate implementation is advised.

Once the slant range has been calculated, the conversion to spatial coordinates is visualized as the geometric relationship between the transmitter (t subscript) and receiver (r subscript):

$$\text{slant range} = [\,(x_t - x_r)^2 + (y_t - y_r)^2 + (z_t - z_r)^2\,]^{\frac{1}{2}}.$$

Because the error in the depth parameters are small compared to the horizontal parameters, the slant range is projected onto the xy plane prior to the adjustment (Figure 1.3):

$$\text{horizontal projection} = [(\text{slant range})^2 - (z_t - z_r)^2]^{\frac{1}{2}} = [(x_t - x_r)^2 + (y_t - y_r)^2]^{\frac{1}{2}}.$$

The left half of the equation is calculated in *xpread*, $CRANS(NTR, NPOS) = (S^2 - D^2)^{\frac{1}{2}}$ where NTR = the number of transponders, $NPOS$ = the number of *FLIP* positions, S = slant range between a position and a transponder, D = transponder depth - *FLIP* depth, and passed into *xpfil* as an array $HRAN$ which is redefined within a loop as a variable HH. The right half of the equation is calculated in *xpfil*, $RNGEC = \sqrt{(XDIFF)^2 + (YDIFF)^2}$, where $XDIFF = XG - XF(NDAT)$, $YDIFF = YG - YF(NDAT)$, (XG, YG) are the xy positions being interated, (XF, YF) are the fixed positions indexed over $NDATA$ data points, and the error ERR, summed over the number of fixed positions, is $(RNGEC - HH)^2$.

The adjustment is calculated using the steepest descent method to minimize the error by following the mean squared error gradient to a minimum. For known transponder positions and the x-direction, the perturbed position is:

$$XG = \sum_{NDAT=1}^{NDATA} XG + h\ ERR'(NDAT)$$

where h is the step size, and ERR' is the negative derivative of the error function with respect to XG. The y-direction adjustment is calculated similarly. The error derivative expands to:

$$ERR' = \frac{d(RNGEC - HH)^2}{dXG} = 2(HH - RNGEC)\frac{d(XDIFF^2 + YDIFF^2)^{\frac{1}{2}}}{dXG}$$

$$= \frac{(HH - RNGEC)}{RNGEC}\,\frac{d(XG - XF)^2}{dXG} = RATIO*XDIFF$$

where $RATIO = (HH - RNGEC)/RNGEC$, and the constants are absorbed by the step size h.

Simulations. Several simulations of the array navigation system were conducted to examine the sensitivity of the estimated transponder positions to errors in travel time measurements and initial positions. The spatial configuration used closely resembles the experimental set up of the September sea test as shown in Figure 1.4. The transponders and *FLIP* were initially assigned to known positions with determined slant ranges as shown. Two simulations were conducted to illustrate the transponder position response to errors; two other simulations were conducted to show the effect of transponder and *FLIP* positional errors on the estimated array positions and are presented in [Sotirin and Hildebrand, 1989]. The result of the first simulation was the 3D error surface for various parameters. The second was a Monte Carlo simulation of the initial transponder positions. The simulations provide an understanding of the

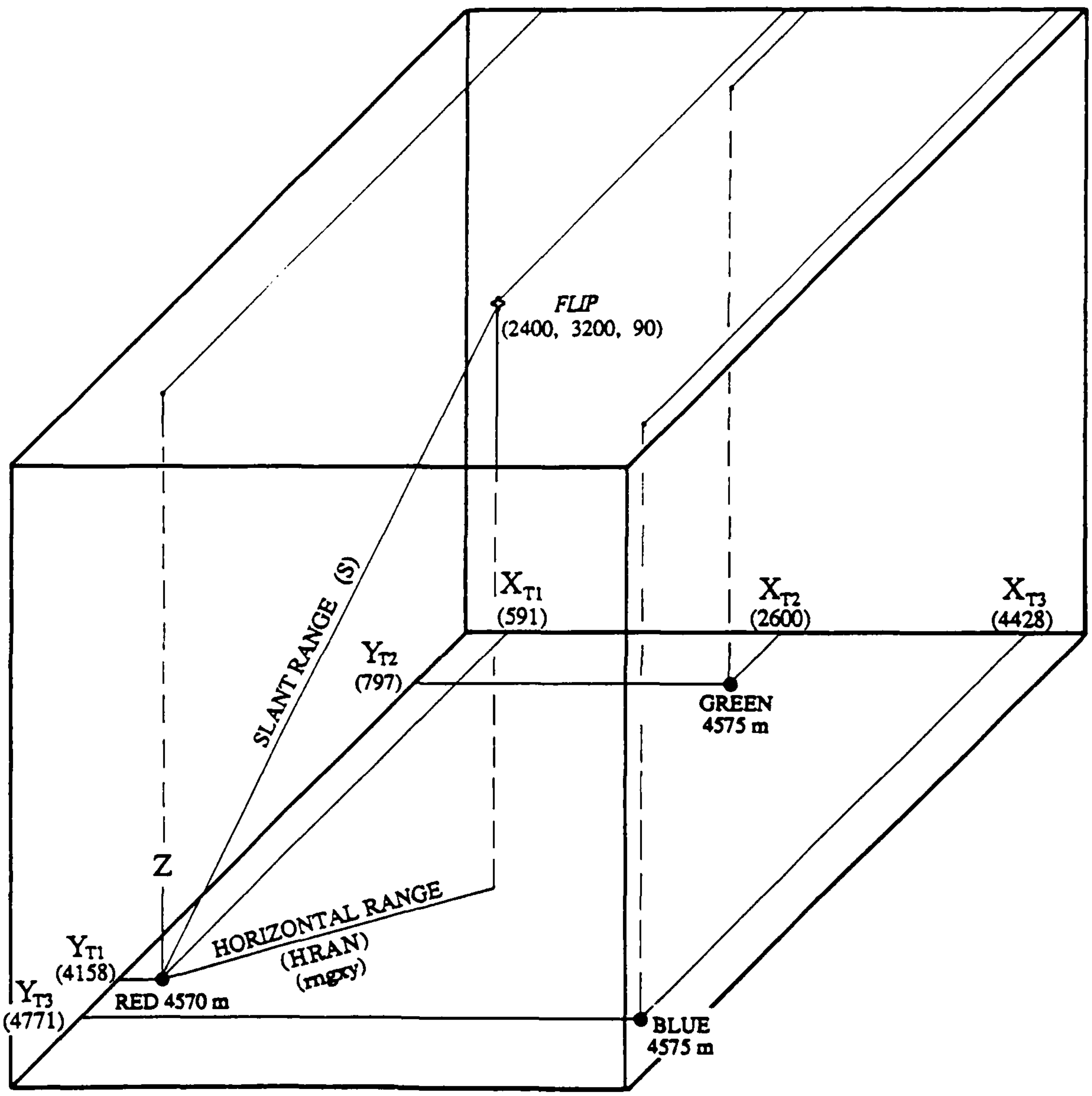

Figure 1.3 Navigation Overview. The horizontal projection is estimated first by using the measured slant range and depths (*HRAN*) and then by using the initial xy positions (mgxy). The initial xy positions of the transponders (T1=red, T2=green, T3=blue) and *FLIP* as measured by a GPS fix are notated in meters on the axis, and beneath the transponder for depth.

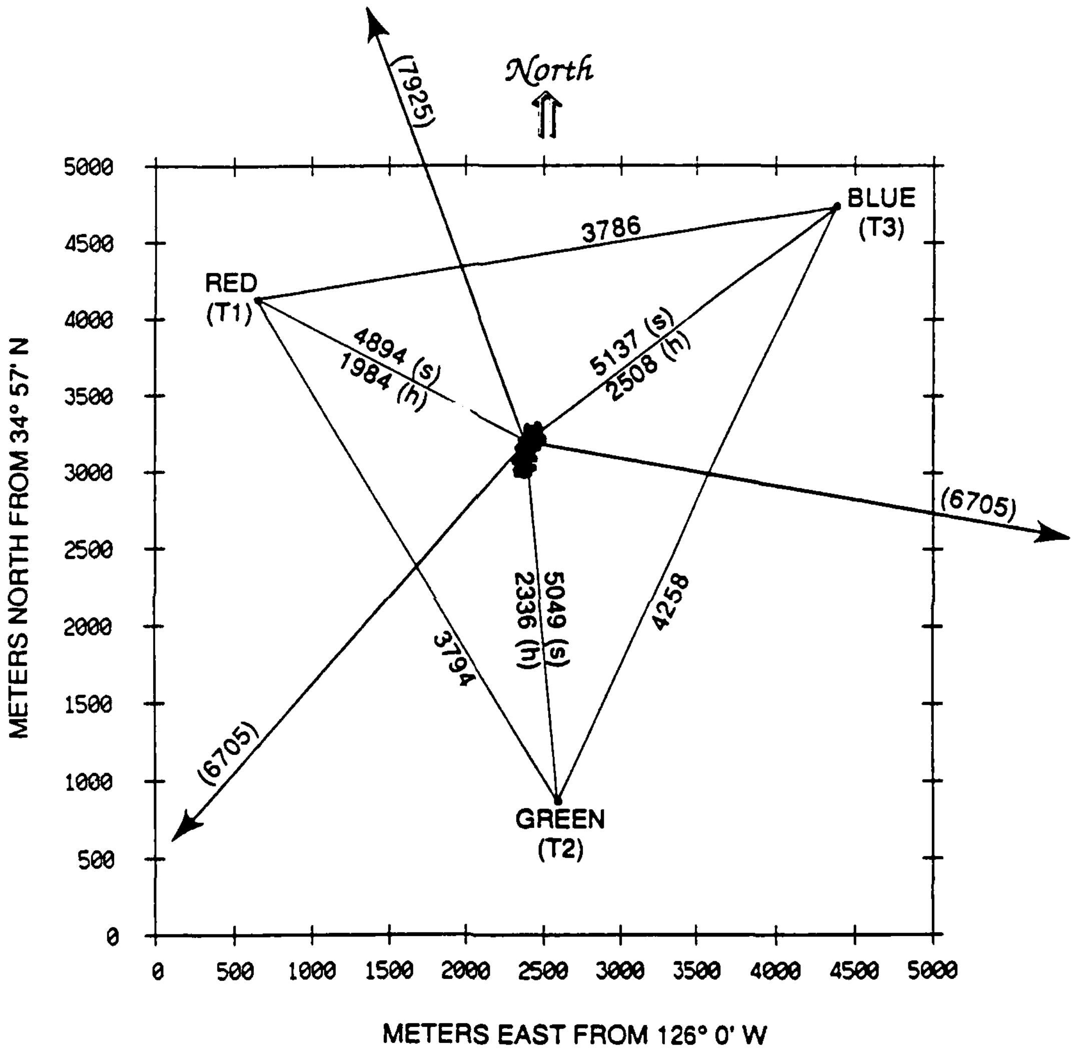

Figure 1.4 Spatial Configuration for Navigation During the September 1987 Sea Test. The estimated xy positions of the 3 fixed bottom transponders and of *FLIP* measured hourly over 18 days are shown in plan view. The mooring lines are represented by arrows, and the slant range (s) and horizontal projection of the slant range (h) are indicated in meters from an arbitrary *FLIP* xy position of (2400,3200) m. Transponder baseline distances are also indicated in meters.

effect of the unconventional survey data.

To examine the error surface for the least squares configuration used in the previous section, specific model parameters were perturbed systematically. The error is the rms value of the differences between the slant ranges based on the travel time measurements corrected for sound speed deviations (Eq. 1.1 dividing through by c) and those calculated from the spatial positions of the transponders and *FLIP* which were output from the least squares filter. Ideally this produces a single well-defined minimum for which the optimization method searches. This is not the case in many real applications however, and the possibility of local minima and/or a broad global minimum should be investigated. The inverted error surface for perturbations in the horizontal positions of the blue transponder is shown in Figure 1.5. It exhibits a narrow channel of local minima which appear as a ridge plotted as the negative logarithm of the error against the perturbation amplitude in x and y position. The ridge is orientated perpendicularly to the *FLIP*/transponder range direction indicating that the azimuthal component is not well constrained. Perturbing the horizontal positions of the other transponders produced similar results.

The system displays a sensitivity to initial positions which is shown to be an artifact of the limitations in the *FLIP* 'survey' discussed previously and of the structure of the error surface. The direction in which the transponders are moved is constrained to nearly parallel to the *FLIP*-transponder baseline (Figure 1.6). This occurrs because the distribution of *FLIP* positions shown in Figure 1.4, provide adequate range information but minimal azimuthal information regarding the transponder positions as was indicated by the error surface. There are no sure techniques for locating a global minimum in the company of local minimina. Without independent positional data corroborating the results, either Monte Carlo techniques would have to be incorporated in the initial position of each transponder, or knowledge of the error surface would have to be employed as a constraint (search range/azimuth space rather than xy space). Fortunately, the initial estimate of the transponder positions were acquired from accurate GPS fixes and the resulting error in GPS positions compared to the estimated *FLIP* positions at corresponding times had an rms value of only 10 m. Thus the estimated transponder positions were declared adequate.

II. Array Travel Time Acquisition.

Travel time measurements were acquired by interrogating three bottom mounted transponders from FLIP and detecting their replies at the navigation receivers distributed across the 900 m aperture array. There are 24 navigation receivers located at ± 3.75 m from each processor which are separated by 75 m (Figure 2.1). Due to bandwidth constraints, data from 12 of the navigation receivers (one/section) were decimated and recorded during the September 1987 sea test. The data bit stream from transmit time of the interrogation pulse to receive time of the reply at the array was reconstructed during post-processing for each navigation receiver and the travel times calculated.

The navigation timing was based on a 16 bit clock driven at a 1 KHz rate which initiated the navigation sequence. This clock is referred to as the hardware clock to differentiate it from the real-time clock (local time) and the Greenwich Mean Time (GMT) clock. The timing is illustrated in Figure 2.2 and described below. A transceiver located at the bottom of FLIP transmited a series of 65536 ms transponder sequences (Figure 2.2a). A transponder sequence consisted of 4 interrogation pulses 10 ms long at 10 s intervals beginning at a 16 bit clock rollover followed by a 35.536 s silent interval (Figure 2.2b). The first three pulses were at the unique transponder interrogate frequencies of the bottom transponders (10, 10.5 and 11 KHz). Upon receiving an interrogation pulse the bottom transponders replied with a 3 ms pulse at 12 KHz. The turn around time of the transponders is signal to noise dependent, however due to the strength of the transmitted signal, the delay was assumed to be less than 1 ms. The fourth interrogation pulse transmitted by the FLIP interrogate transceiver was at 12 KHz which simulates a bottom transponder

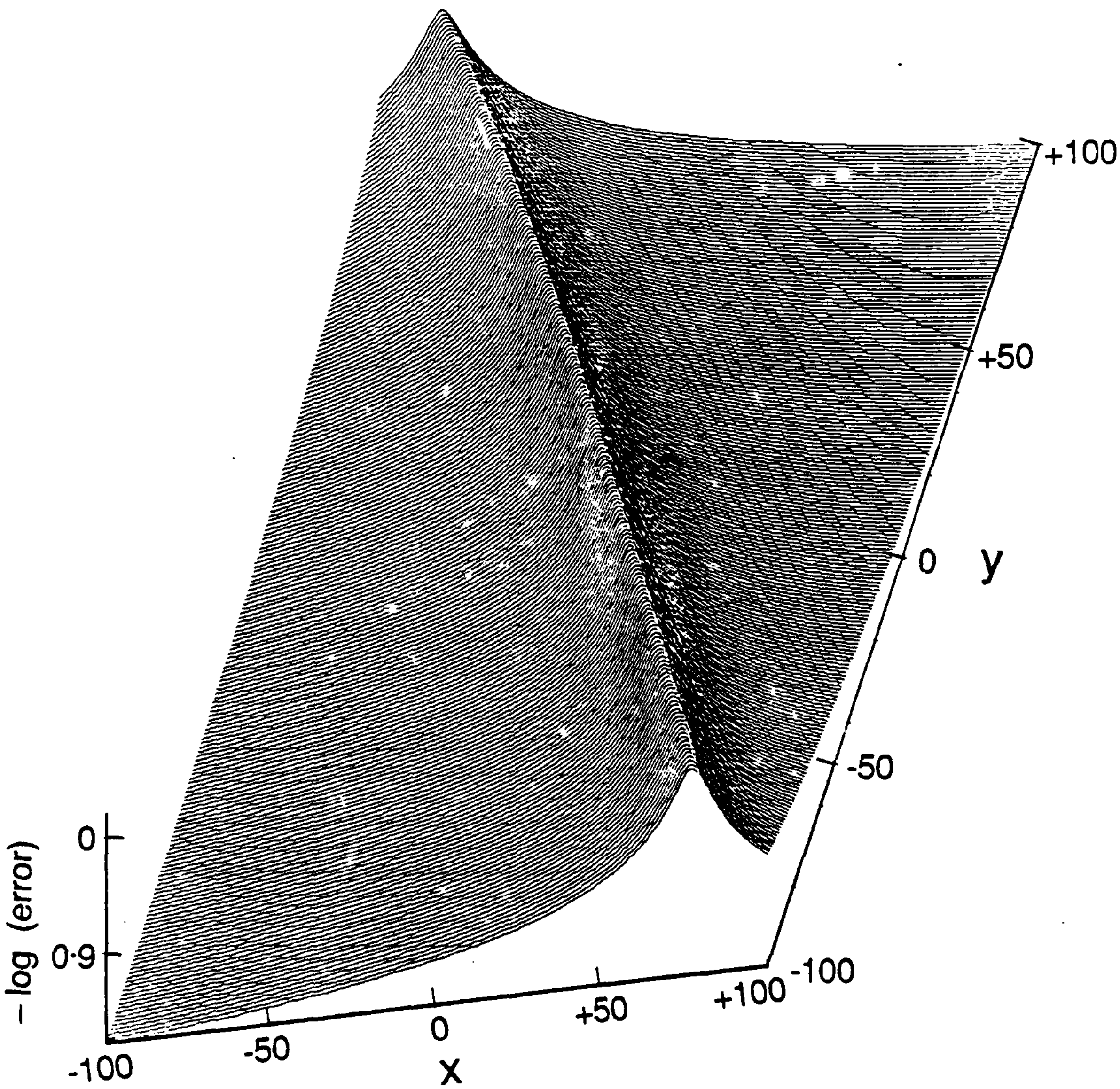

Figure 1.5 **3D Error Surface.** The estimated positions of *FLIP* and the transponders were considered known, selected parameters were perturbed and the resulting error calculated. In this example, the blue transponder x and y positions were perturbed in 1 meter increments to ±100 m from the original known position which is plotted at the center. The z axis is plotted as the negative logarithm of the error to allow visualization of the minima.

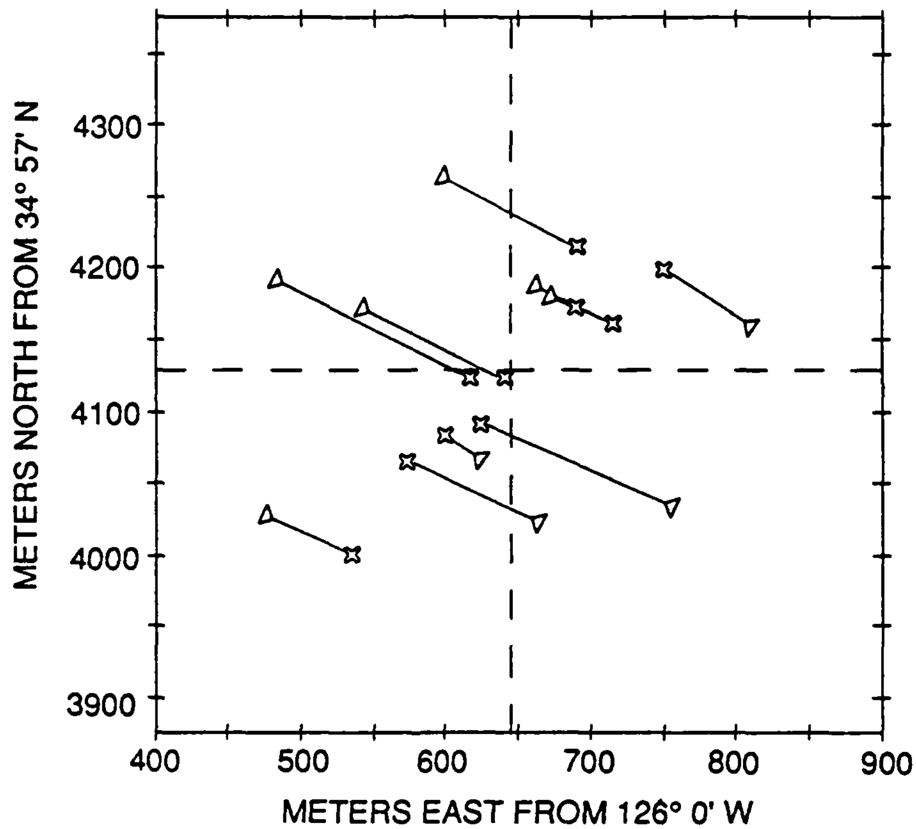

Figure 1.6 **Monte Carlo simulation of initial transponder positions.** This simulation illustrates the constraints in search direction placed on the least squares iteration by the spatial configuration of the survey points. The azimuthal position of the transponders is not accurately determined by the *FLIP* data set used. The transponders tend to move along the transponder to *FLIP* direction. The initial transonder position is notated by a triangle, the final position by a cross.

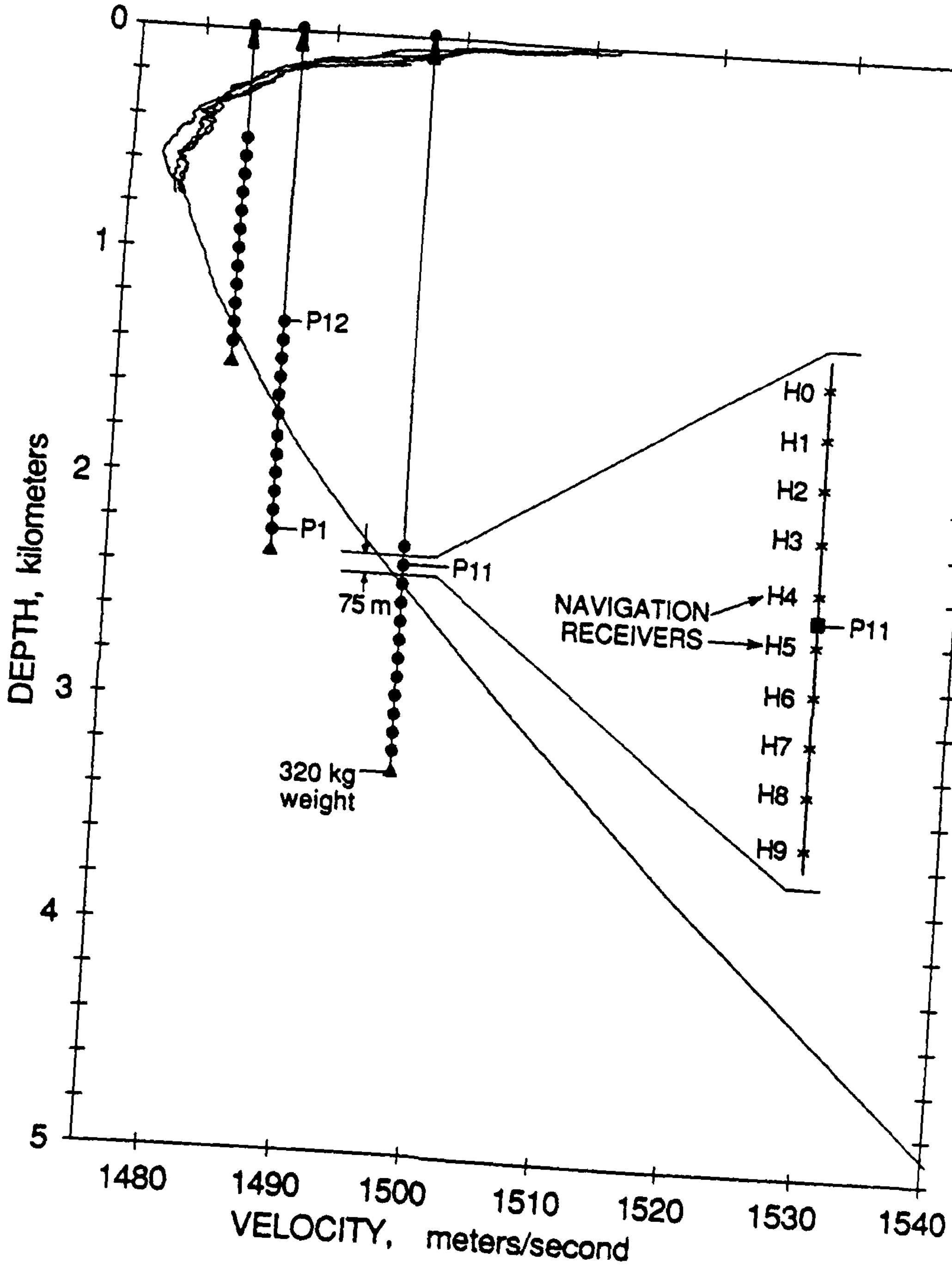

Figure 2.1 **Array Navigation Receivers.** The array was deployed at three nominal depths during the September 1987 experiment. The 12 indentical array sections are each 75 m in length, with 10 receiving hydrophones spaced at equal increments. The location of the navigation receiving hydrophones is illustrated for one section. The data from one navigation receiver per array section (H5) was recorded during the experiment. The background curves represent the CTD sound speed profile and 3 near surface profiles calculated from XBT data.

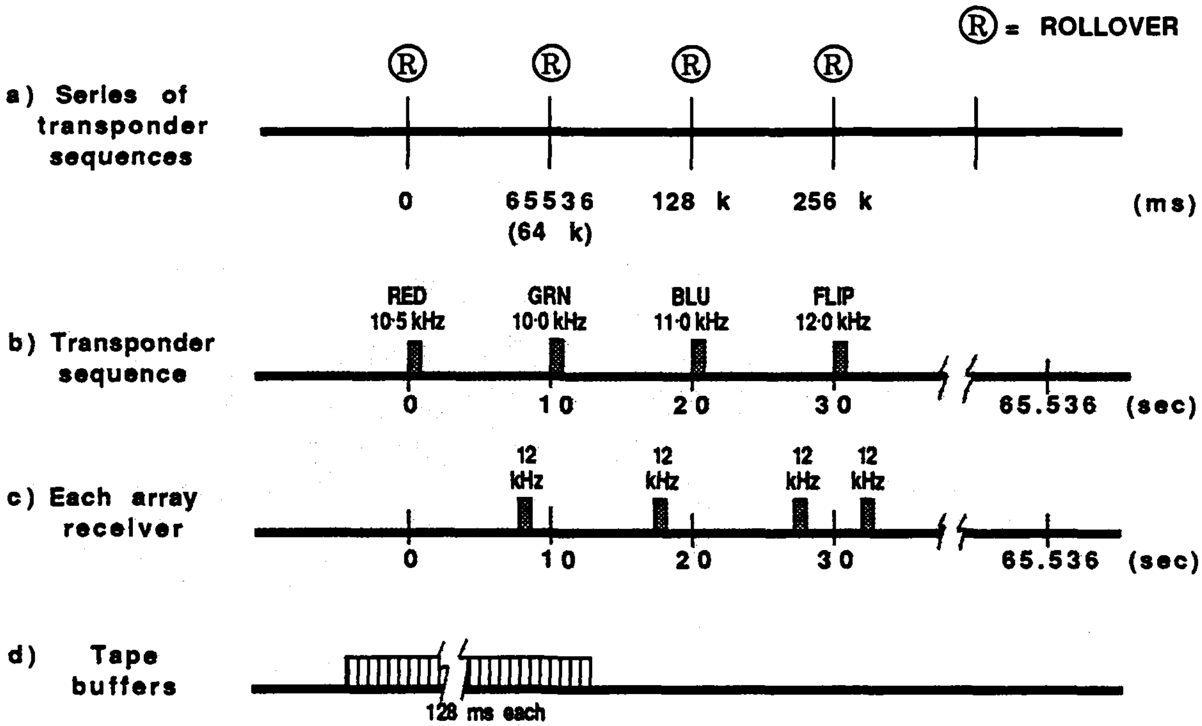

Figure 2.2 **Navigation Timing Diagram.** The timing associated with the array navigation system is illustrated. (a) The transponder interrogation sequence is initiated every 65536 (64 K) ms by a 16-bit hardware clock rollover (R). (b) Each transponder sequence consists of 4 transmissions at unique frequencies issued from a transceiver mounted on the bottom of *FLIP* (90 m depth) spaced 10 s apart. (c) The array receives the 12 kHz transponder replies and the 12 kHz pulse from *FLIP*. The time of arrival of a transponder reply reflects the time it took the acoustic pulse to travel from *FLIP* to the transponder to the array receiver. The time of arrival of the *FLIP* 12 kHz pulse corresponds to the distance from the array receiver to *FLIP*. (d) The hardware clock rollover and 12 kHz arrivals may occur anywhere within a 128 ms tape buffer which complicates the data extraction procedure.

reply. The array therefore received four consecutive 12 KHz reply pulses about once a minute (Figure 2.2c). The navigation receivers were capable of detecting a 12 KHz signal 6 dB below the noise level in a 200 Hz band. The binary output of the detector was sampled at 5 KHz and decimated by 2 to provide a continuous time series consistent with the 5 bits per processor every 2 ms allowed for navigation data within the specified data format. Thus the data from 12 navigation receivers located 3.75 m below each processor (H5 in Figure 2.1) in the array were multiplexed in with the low frequency acoustic data, transmitted to the surface and recorded. The interrogation sequence was synchronized with the timebase in the array and the initiation time of the sequence (as indicated by the rollover of the hardware clock) was sampled every 128 ms and recorded on the tape. The tape format showing the placement of the navigation data is illustrated in Figure 2.3. The navigation data time series are reconstructed from this recorded data by the method described below.

The structure of the data on tape was not optimized to facilitate extraction of the navigation bit stream. The header containing the timing information appears in the first 8 words of each tape buffer which contains 128 ms of data. Most of the 6280 16-bit words within a buffer are assigned to low frequency acoustics rather than navigation as seen in Figure 2.3. To reconstruct the sampled time series output from each navigation receiver, the 5 bits/processor must be extracted and stored. The major programming effort was in initializing and incrementing pointers and in error checking. A description of the main program and subroutines and a sample run will be found in Appendix B. The extraction of the data begins with the data buffer containing the hardware clock rollover. The rollover may occur anywhere within the data buffer. The program examines each buffer header for the hardware clock rollover. This is determined by testing between consecutive clock times for a negative difference, and linearly interpolating within the buffer for the correct frame. For example, if the hardware clock in buffer 1 is $hwc_1 = 65430$ and the hardware clock in the next consecutive buffer is $hwc_2 = 22$, then the difference ($hwc_2 - hwc_1 = -65408$) is negative and the clock rollover occurs within buffer 1 at frame 53; the data stored would begin at frame 0, buffer 1. The time difference from the beginning of the buffer to the rollover is stored in a parameter called *start* and passed to the subroutine *navloc* which calculates the travel time. Once the rollover is identified, the navigation data associated with each processor is masked off and stored as the 10 most significant bits in a 16-bit word (*fillnav*).

The travel time calculation is a simple difference once the time of the transponder reply is determined. To locate the reply, the data is treated as a time series of 0.4 ms bits, and a correlation between the time series data and a replica transponder pulse is initiated as a matched filter detector. Due to inconsistencies in receiver detection threshold and noise level, each receiver is assigned an individual replica pulse length. Temperature sensitivity of the capacitors and high failure rate of the inductors in a phase matching tuned filter caused the mismatch in the detection threshold of the individual receivers. This variation in receiver error is apparent in the positional error distribution shown in Figure 2.4, in which the only hardware or processing difference is in the navigation receiver itself. Additional variation in receiver signal to noise level of the incoming reply could have been caused by the 12 KHz beam pattern of the individual elements. Each element consisted of two hydrophones wired in series with a spacing between 8 and 9 cm such that at 12 KHz, a notch in the beam pattern appears between 46° and 52° from broadside. From Figure 1.4, the arrival angle of the transponder replies when the array was nominally at 400 m was between 58° and 65° from broadside and signal to noise was high enough to be detected; at deeper depths, however, this is potentially a problem. The rms error used in the distribution calculations accompanies each receiver position estimation described in Section III. The transponder reply is a CW pulse, so the correlator is implemented as a moving adder, with a detection defined as the first occurrance of a normalized correlation amplitude greater than or equal to 1.0 within a valid data window.

The window was installed due to excessive noise levels and the interference seen in the return of a single navigation receiver from the hardware clock rollover as shown in Figure 2.5. The window must be determined prior to program execution. An option in the program *harrynav* (-p) will print out all bits set by the navigation receiver, packed into the least significant 10 bits of a 16-bit word (for a maximum of 3FF) to allow the user to determine the window parameter. An example is seen in Figure 2.6: the return for

TAPE (≈23.5 minutes):

BUFFER 1 (6280 16-bit words)
BUFFER 2
...
...
BUFFER 11000

BUFFER (128 ms):

8 Word Header
Frame 0 (98 words)
Frame 1 (98 words)
...
...
Frame 63 (98 words)

FRAME (2 ms):

Frame Sync Word (EB90x)
Telemetry Module Word
Processor 01 Data (8 words)
Processor 02 Data (8 words)
...
...
Processor 12 Data (8 words)

8 Word Header:

wrd 0	wrd 1	wrd 2	wrd 3	wrd 4	wrd 5	wrd 6	wrd 7

Word 0: **Hardware clock** - a 16 bit binary clock which is synchronized with the array and the transponders. *It is used as a second order time approximation* with a millisecond resolution, turning over every 65536 milliseconds. Since it is read on interrupt from the DMA controller AFTER the data is read into the tape buffer, the value must be corrected to reflect the time at the beginning of the buffer (128 - 0.016*(32-NPROC) ms).
Word 1: **Sequence number** - a 16 bit counter associated with each buffer. It is set to zero when the tape driver is initialized and indicates whether any buffers are missing from the tape.
Words 2 and 7: **GOES Satellite Receiver Clock** - this is the clock which allows synchronization of the array data timebase with the real world. The clock is read in as 8 BCD digits: hours (1), minutes (2), seconds (2), and milliseconds (3) on interrupt from the DMA controller AFTER the data buffer is read in. Word 2 contains the first 4 digits (hours, minutes and 10's of seconds); Word 7 contains the second 4 digits.
Word 3: **Unread buffers (MSB) and buffer number (LSB)** - housekeeping words used to monitor the tape driver.
Words 4, 5 and 6: **Real-time clock** - free running time of day clock which is not synchronized with any of the other clocks. It is used for a first order time and date approximation with a one second resolution.

PROCESSOR DATA

	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8	bit 9	bit 10	bit 11	bit 12	bit 13	bit 14	bit 15
Word 0	←------					Phone 9						------→	←----			
Word 1	----			Phone 8				----→	←----				Phone 7			----
Word 2				----→	←------					Phone 6						------→
Word 3	←------					Phone 5						------→	←----			
Word 4	----			Phone 4				----→	←----				Phone 3			----
Word 5				----→	←------					Phone 2						------→
Word 6	←------					Phone 1						------→	←----			
Word 7	----			Phone 0				----→		Navigation				PID		

Word 4	month	day
	0xF	0x1F
Word 5	hour	minute
	0x1F	0x3F
Word 6		seconds
		0x3F

Figure 2.3 **Data Tape Format.** The tape consists of 11000-12000 buffers, a buffer includes an 8 word header and 64 98-word frames, the header stores relevant timing and error information, a frame contains a frame sync word, a frame counter, and twelve 8-word processor data groups, a processor data group has ten samples of the low frequency acoustic field, 5 navigation bits and a 3 bit processor ID. The information relevant to the navigation includes the 8 word header, frame sync word, telemetry module word, 5 (0.4 ms sample rate) navigation bits (bit 8 occuring first in real time) and a 3 bit processor ID (PID).

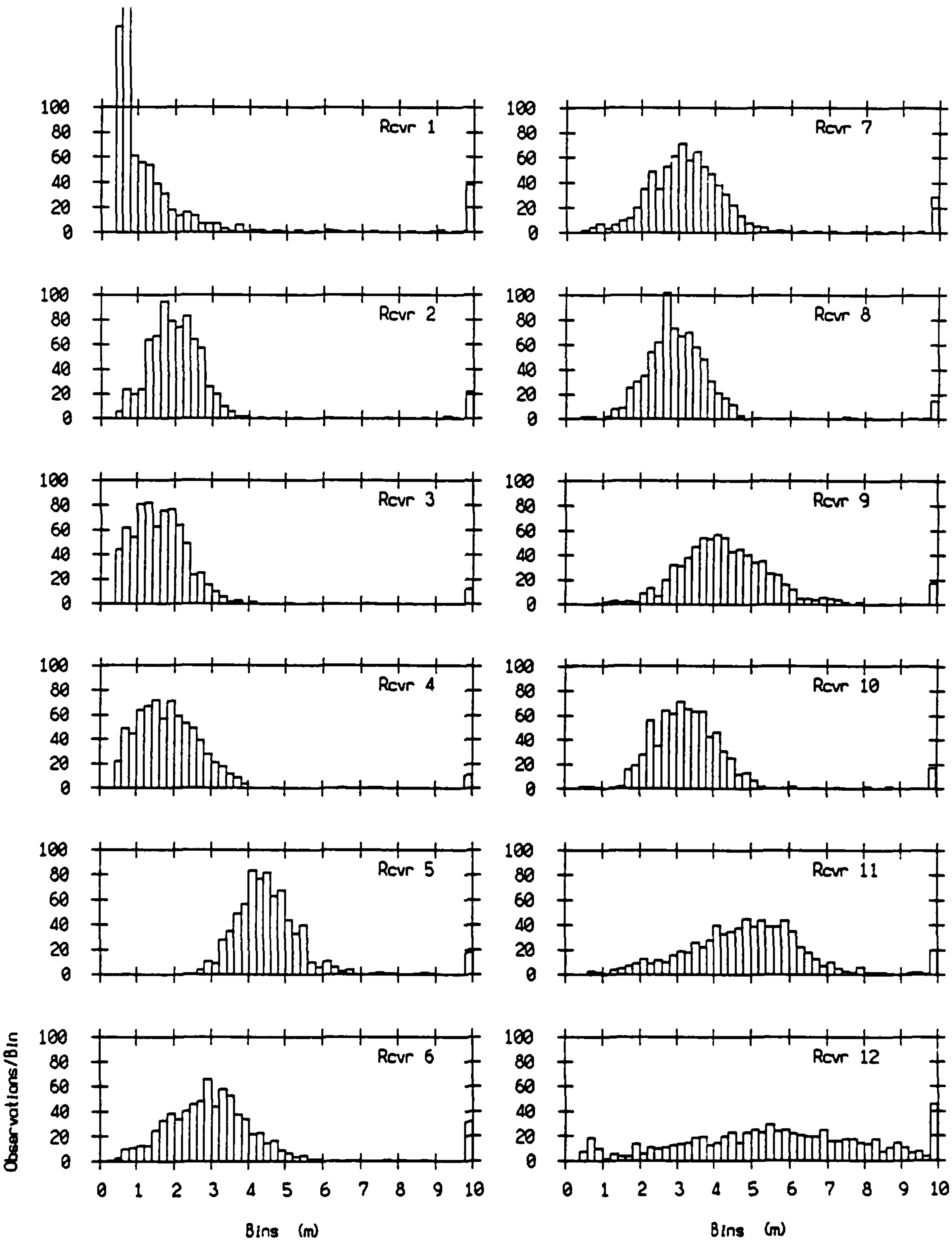

Figure 2.4 **Receiver Error Distribution.** The detection threshold mismatch in the array receivers causes a variation in receiver positional error distribution. This rms error is an output of the program *flpnav* described in Appendix C.

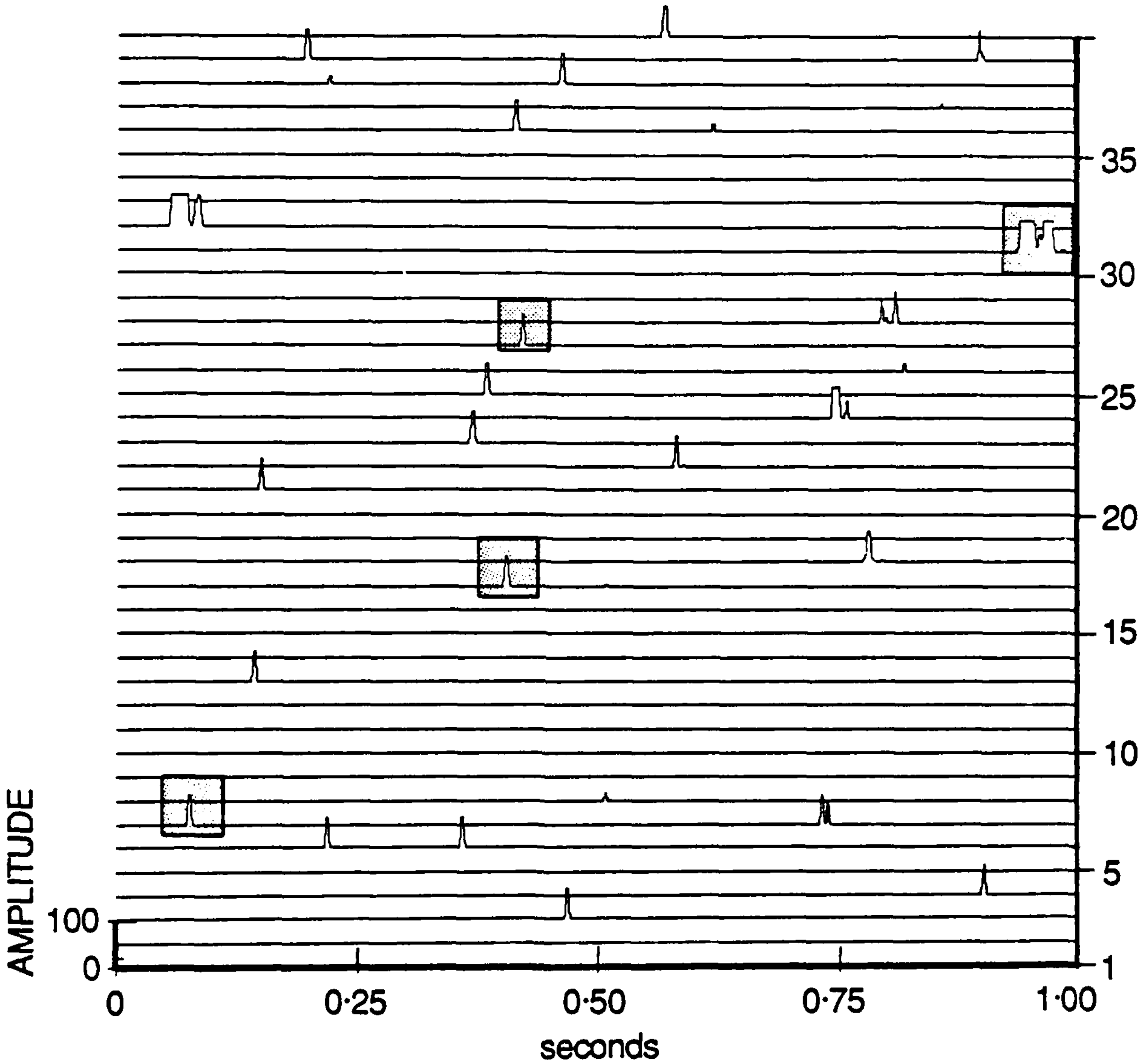

Figure 2.5 **Correlated Output from a Single Receiver.** A single receiver should ideally display only 4 detections during a transponder interrogation sequence. Shown is a 40 s time series of the correlated output of processor 3. Each line represents one second of data, a signal is detected as a transponder return if the amplitude is above the zero level of the line above it. The noise and interferring signals warrented that only data within a valid data window be considered. The desired detections are indicated by the stipled boxes.

processor 2 is clean and the transponder reply is quite clear. The last 4 bits of word 1503 are set followed by 8 bits in word 606, for a total of 12 sequential bits set. The words have a maximum of 10 bits or 4 ms. The valid data window length is a minimum of 100 ms and varies with the transponder. Only the leading edge of the window is specified in the window parameter input file *window.dat*. An estimate of the leading edge of the window for a particular receiver p may be obtained by multiplying the first word of the transponder reply W_p by 4 and subtracting 184 ms. A window parameter for each processor and each transponder must be specified, 48 total. The algorithm for computing the time of arrival of the leading edge of the reply is $T_a = W_p * 4 + (10 - B_p) * 0.4 - start - c$ where W_p is the first word, and B_p is the number of bits set in the first word, start is the time difference between the beginning of the data buffer and the hardware clock rollover as described earlier (printed in line 1 of Figure 2.6 as Start: 14), and c is a constant = 134.68 ms; e.g. W_p = 1503, B_p = 4, T_a = 5865.72 ms which is the result shown in Figure 2.6 for Roll: 1, Xponder: 1 Start: 14, Proc# 2 in the Figure. The constant c is a sum of the delays through the array system in ms determined by laboratory tests to be 7 + (128 - 0.016 * (32 - NPROC)) where NPROC is the number of processors in the array. The 7 ms is due to data buffering in the array hardware and the remainder is due to buffering within the driver for the magnetic tape. This buffering delay was discovered later to be in error by 112 μs and should have been calculated as (128 ms/frame - [125 (words/frame) - 2 (header words) - NPROC * 8 words/processor] * 0.016 ms/word).

The result of the correlation is seen as a series of returns across the array for each of the transponders as shown in Figure 2.7. One second of data for each processor is plotted during each transponder reply with the deepest processor P1 plotted on the bottom. The reply from the bottom transponders (Figure 2.7a, b and c) appear at the deepest processor first and arrive sequentially at the shallower receivers as the pulse travels up through the water column. The pulse from the last transponder (Figure 2.7d) travels down from *FLIP*. The squared amplitudes are normalized to 1.0 and plotted such that a value of 1 will be slightly above the zero level of the next processor. The noise and interference mentioned earlier is also evident; processor 6 represents a particularly noisy time series and interferring signals are seen travelling up and down the array, e.g. between 0.6 and 0.7 s on the plot of transponder 3. Each valid data window is marked with "x's"; the window length for the first and last transponders is 100 ms, but was increased for transponders 2 and 3 to 250 ms to enable detection of the transponder error modes illustrated by the green transponder (transponder 2) in Figure 1.1. The time of arrival of the detected reply may be estimated from the plot by adding the time indicated by the plot to the number of ms notated at the top of the plot and subtracting a constant c = 134 ms defined previously. The travel times are converted into slant ranges by assuming a constant sound speed of 1500 m/s and written into ascii files for use in the spatial position calculation computed in the next program.

III. Array Spatial Localization.

The navigation algorithm for the array elements is virtually identical to that described for the transponders once the initial array X-Y position is determined, except that the transponders are considered stationary. The element to transponder slant ranges corrected for sound speed, the depth of the transponder installed on FLIP and the adjusted transponder locations constitute the data required to navigate the array. The array positions are iterated to achieve the best fit to the data is a least squares sense. A description of the main program and subroutines and a sample run will be found in Appendix C. Array element relative location accuracies of a few meters are achieved by this method.

The array is not moored but hangs vertically from *FLIP* under 320 kg of tension; its horizontal range of motion is significant during a one minute time interval. Consequently, receiver locations are iterated using a single time slice of slant range data. The receiver slant range is the path from *FLIP* to the transponder to the receiver; *FLIP* slant range is simply the path to the transponder. The slant ranges are

```
Roll: 1, Xponder: 1 Start: 14
Proc# 1; 5822.120117 8733.179688 8
Proc# 2; 5865.719727 8798.580078 8
Proc# 3; 5909.319824 8863.979492 8
Proc# 4; 5953.319824 8929.979492 8
Proc# 5; 5998.120117 8997.179688 5
Proc# 6; 6042.520020 9063.780273 7
Proc# 7; 6087.319824 9130.979492 4
Proc# 8; 6132.520020 9198.780273 4
Proc# 9; 6177.719727 9266.580078 6
Proc# 10; 6222.919922 9334.379883 7
Proc# 11; 6269.719727 9404.580078 5
Proc# 12; 6314.120117 9471.179688 7

Processor# (1) 8
 4 8   6 c   8 8   9 8c   10 c   12 8
 13 4   18 4   23 180   25 c   27 80   28 180
-------------------------
1476 80  1477 c  1479 c  1487 80  1489 4  1490 180
1491 c  1492 7  1493 3fc  1497 4  1498 c0  1499 1c4
1501 4  1502 8  1506 84  1508 80  1509 80  1512 180
-------------------------
Processor# (2) 12
876 40  1278 40  1503 f
1504 3fe  1910 80
Processor# (3) 3
177 80  215 4  409 4  492 2
771 80  1038 80  1514 1f  1515 3fc  1651 80  1673 4
1701 4  1791 80  1871 1  1872 200  1876 1  1878 2

Processor# (4) 18
 6 c   23 180   40 c   43 4   52 c0   62 80
 68 1   70 4   72 100   77 84   79 100   87 c
------------------
1389 c  1393 c0  1412 4  1420 100  1424 4  1442 80
1452 80  1456 82  1459 100  1477 8  1486 80  1522 100
1525 1f  1526 3fc  1528 4  1531 8  1534 8  1539 3
1546 80  1548 8  1552 c0  1553 84  1558 8  1560 4
1561 104  1571 80  1573 1c0  1577 80  1578 80  1583 80
1593 80  1596 4  1597 84  1602 180  1606 80  1610 20
1611 4  1614 88  1622 1  1624 4  1628 6  1629 100
--------------------

Processor# (5) 22
1536 7  1537 3f8

Processor# (6) 16
--------------------
1533 1c4  1534 e4  1535 1c6  1536 1c4  1537 c0  1538 c0
1539 87  1540 e0  1541 84  1542 1c4  1543 4  1544 4
1545 ee  1546 4  1547 3  1548 3ff  1549 180  1550 6
1551 4  1552 e4  1553 c  1554 86  1555 6  1556 80
------------------
Processor# (7) 7
638 100  1559 3fe
Processor# (8) 21
1570 7f  1571 300
1846 1f  1849 40
Processor# (9) 17
1581 f  1582 3c0
Processor# (10) 19
1592 1  1593 3ff
1594 200  1787 203  1788 3e0
Processor# (11) 20
1604 f  1605 380
Processor# (12) 5
 23 6
 57 4  110 4  127 80  172 80  212 4  266 4
397 80  445 80  508 80  522 80  549 4  592 8
722 4  810 80  875 180  899 80  905 80  952 4
967 2  983 20  1057 6  1063 4  1139 6  1250 80
1277 4  1285 4  1289 4  1296 40  1307 80  1361 80
1411 180  1548 80  1608 80  1613 80  1615 7  1616 3fe
1701 4  1713 80  1763 78  1765 ef  1766 3c0  1784 4
1836 40  1905 60  1992 4  2019 80  2040 80  2075 4
2096 c  2145 80  2148 4  2162 80  2183 4  2191 80
2195 4  2224 c0  2229 8  2260 4  2287 c  2294 80
```

Figure 2.6 **Raw Navigation Data.** Raw data output for one transponder return. Some of the receivers are very noisy and the dashes indicate that data have been deleted for clarity. The processed output is listed first indicating that this is the first rollover, the first transponder and that the difference between the hardware clock rollover and beginning of the buffer is 14. The processors are numbered 1 to 12 with 1 located at the deep end of the vertical array. The first number is travel time in ms, the second is slant range in m assuming a constant sound speed of 1500 m/s, and the third is an indication of the strength of the return. The raw data is printed below, listing the processor number and ID, followed by the data word number and bits set of the hardware receiver output. Each word represents 4 ms or 10 bits for a maximum of 3FF if all 10 bits are set.

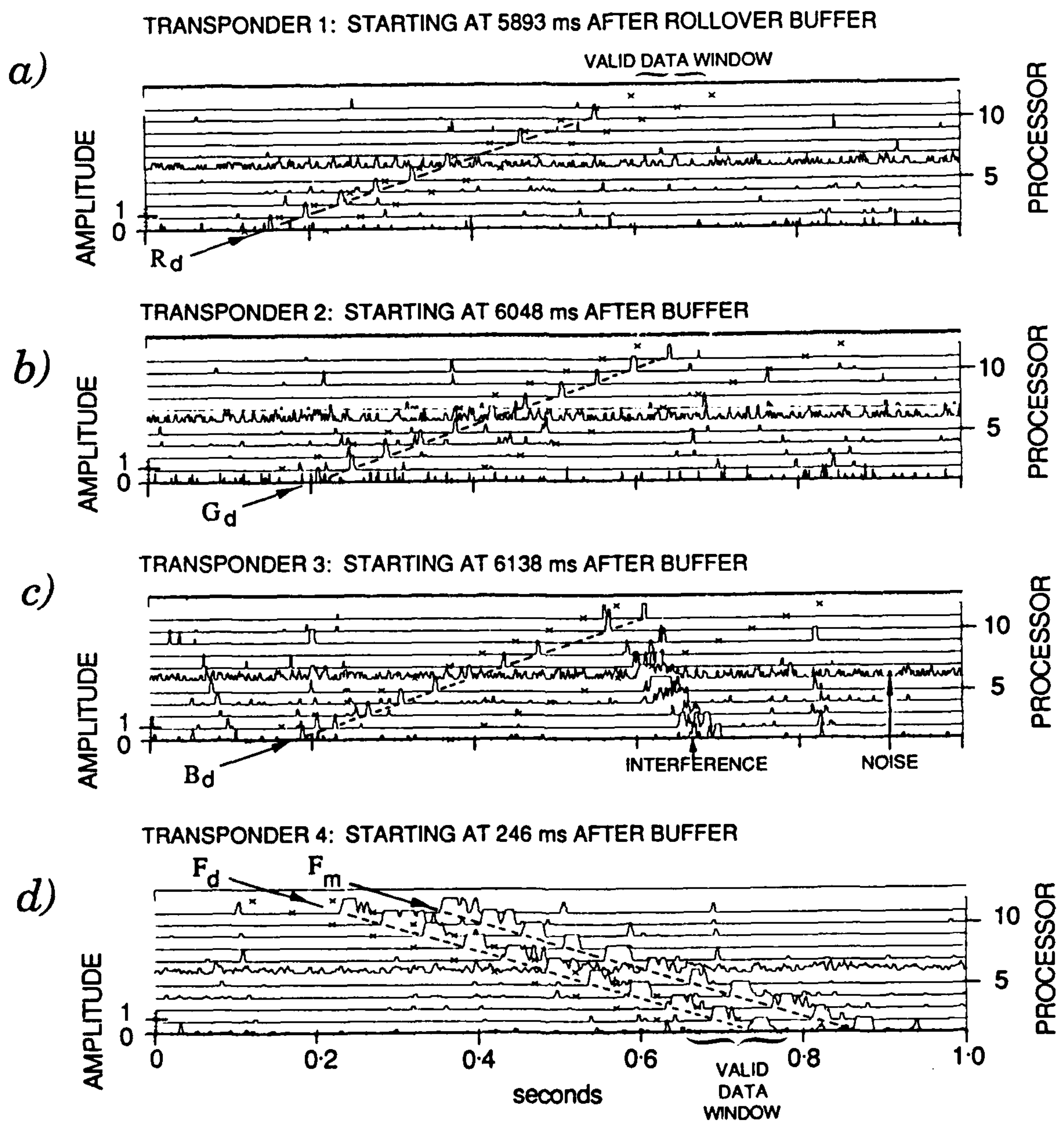

Figure 2.7 **Transponder Reply Detection.** The correlator output for each processor is plotted for 1 second around the time of each of the transponder replies. The top processor plotted is the top processor in the array. The valid data window is indicated by the small x's. The bottom mounted transponder replies are evident as upward traveling pulses, arriving at the bottom of the array first. (a) T1 (red) transponder reply arriving between 0.15 and 0.55 for processors 1 and 11 respectively; (b) T2 (green) transponder reply; (c) T3 (blue) transponder reply; (d) the 12 kHz pulse transmitted from *FLIP* arrives as a downward traveling pulse with the surface bounce 120 ms behind.

corrected for deviations due to a sound speed profile which differs from the assumed 1500 m/s, as are the receiver depths (which are obtained from the travel time measurement of the *FLIP* transceiver to each receiver), using the harmonic mean described earlier. The noise in the receiver slant range measurements (Figure 3.1) is contributed to not only by the receiver noise as shown in the previous section but by the transponder malfunction described in the first section and by high frequency *FLIP* motion (recall that *FLIP* slant ranges are available only once per hour). Constraints on the difference in receiver depths and slant range measurements from one minute to the next are implemented as user parameters *thresd* and *thres*. If these thresholds are exceeded, the noisy data is ignored and another user parameter *alter* determines which interpolation scheme will be implemented, if any.

Localization of the array receivers now proceeds to the least squares filter. A constant initial xy position is assigned to *FLIP* based on a GPS position as was done during the transponder iteration for the *FLIP* initial position; the initial positions for the first receiver iterated is the estimated *FLIP* position; the initial positions for subsequent receivers is the estimated position of the previous receiver. The *FLIP* slant range is subtracted from the receiver slant ranges leaving the transponder to receiver portion. The horizontal slant ranges are calculated and the xy positions are iterated minimizing the squared error in calculated and 'measured' positions. With the transponder positions fixed, the iteration is confined to *FLIP* and the array receiver positions whose x, y and z positions are output with an associated rms error. The resulting analysis of data processed in this manner is presented in [Sotirin and Hildebrand, 1989].

Acknowledgments

This work was supported by the Office of Naval Research under Contract nos. N00014-87-K-0225 and N00014-87-C-0127. The authors are indebted to the undaunting work of the MPL staff. We wish to thank Vince Pavlicek for engineering expertise, Eric Wolin, Paul Henkart, Dick Currier and the MPL Deeptow Group for software support, Tony Aja, Dave Ensberg and Pam Scott for their assistance in the data processing effort and Jo Griffith for her illustrative talents. The captain and crew of the research platform *FLIP* contributed to the sea-going operation. Special thanks goes to Chris deMoustier for his critical reviews and valuable suggestions.

References

Smith, W., W. M. Marquet and M. M. Hunt, "Navigation Transponder Survey: Design and Analysis", in I.E.E.E. Ocean 75 Record, San Diego, CA, 563-567, Sept., 1975.

Sotirin, B. J., F. V. Pavlicek and J. A. Hildebrand, "Low Frequency Digital Acoustic Array", in Current Practices and New Technology in Ocean Engineering-1988, ed. G. K. Wolfe and P. Y. Chang, American Society of Mechanical Engineers, NY, OED-Vol. 13, 1988

Sotirin, B. J. and J. A. Hildebrand, "Acoustic Navigation of a Large Aperture Array", in preparation, 1989.

Spiess, F. N., "Analysis of a Possible Sea Floor Strain Measurement System", Marine Geodesy, Vol. 9, No. 4, 385-398, 1985.

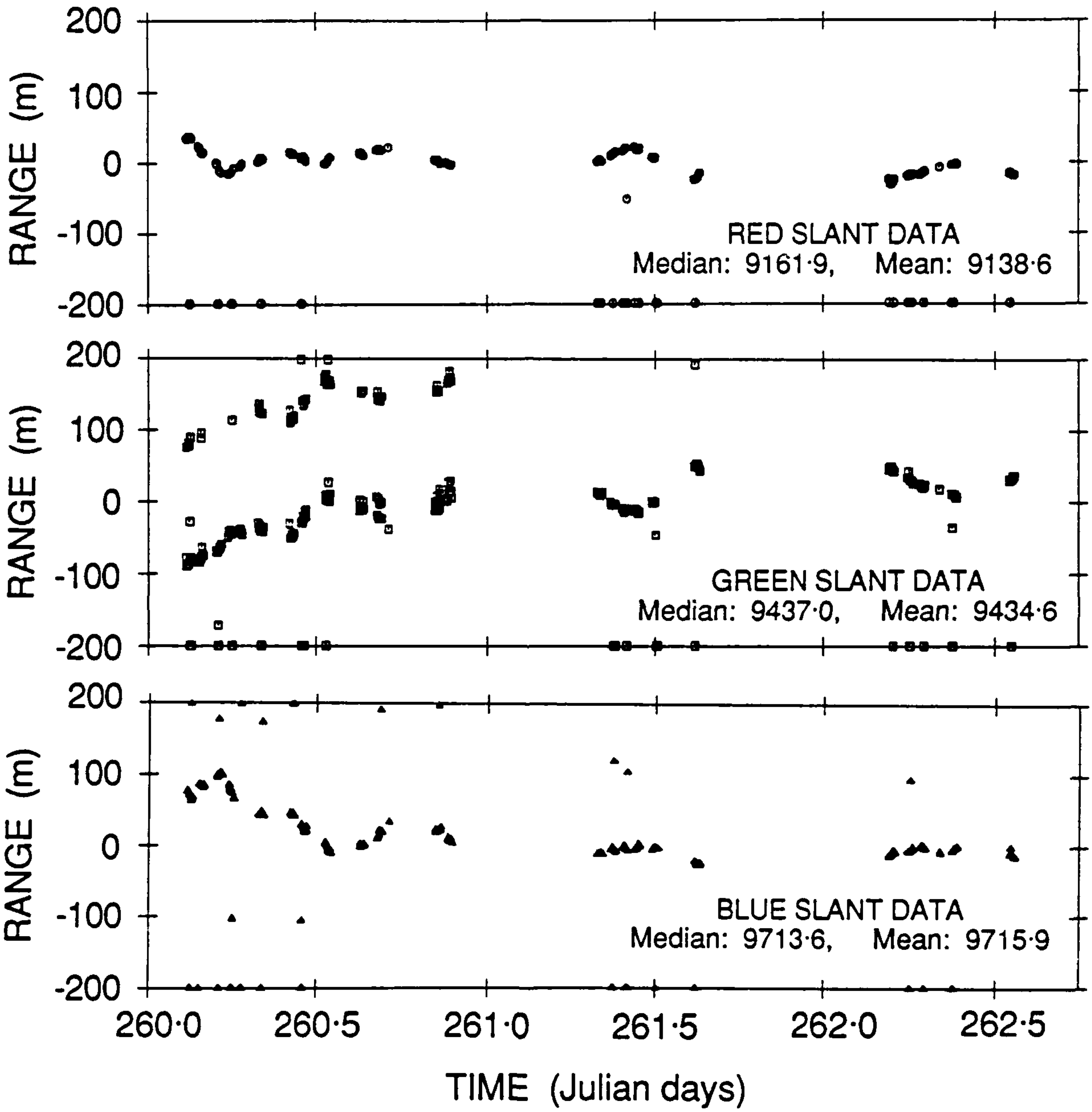

Figure 3.1 **Slant Range Data.** The slant range data for each of the bottom transponders (as detected by the navigation receiver sampled by processor 8) are plotted relative to the median value for a series of nonconsecutive tapes. The data were clipped at ±200 m from the median value. The errors shown in Figure 1.1a for the green transponder are evident as a shadow return 160 m from the direct return. Most of the data appearing at -200 m illustrates a lack of detection due to the navigation of *FLIP* once an hour for several minutes. *FLIP* navigation continues to interrogate the transponders consequently the 12 kHz returns are there, but because the *FLIP* navigation system is not synchronous with the array, the valid data windows filter out most of the unwanted returns.

APPENDIX A. Transponder Localization Program Description.

The tables below illustrate the program flow; the text details purpose of each routine. The structure of the tables reflects the main program and subroutine levels. Each subroutine in level 1 is called by the main program; each subroutine in level 2 is called by the level 1 subroutine to the left. The structure of the text is similar; indentations reflect subroutine level. Any file names beginning with xxx may be specified by the user. The first routine localizes the transponders and is called *XPMAIN*. It is assisted by an initilization program called *XPLOAD*. These programs were originally written for navigating the transponders used in the DEEPTOW program at the Marine Physical Laboratory. A sample run for each program specifing the input and output file structures and program user inputs follows the description of the program.

XPLOAD

Main	**Subroutines**	
Program	Level 1	Level 2
XPLOAD		
	XPCMLD	XPSET
	BASLIN	
	XPCMLD	
	EXIT	

XPLOAD: initializes the program parameters used by the main transponder navigation program XPMAIN. There are three files involved:

XPCOM.DAT defines a common area containing initialized variables. If this file does not exist when the program is initiated, it is created.

xxx.trs contains the X, Y and depth information for each transponder and provides a convenient way of inputting the data. The default name is TRANSPONDER.TRS, read in if a carriage return is input when the user is querried for the transponder file.

xxx.lst contains the X, Y depth and baseline information for each transponder. The user is querried for the name, referring to the 'listing' file.

XPCMLD: is a subroutine which reads or writes to a file depending upon the state of *ldflg*. The file is called *XPCOM.DAT* by default and if it does not exist when XPCMLD is called, it is created.

XPSET: If *xpcom.dat* does not exist, the file is created; program functions include initializing the deep (1500 m/s) and shallow (1500 m/s) sound velocities, logical unit numbers, plotter inputs and clearing the transponder data buffer areas. Some of this information is obsolete eg. sound velocities and plotter inputs.

If the transponder data file *xxx.trs* does not exits, it is created by quering the user. The inputs for each transponder are 'Label, X, Y, Depth, Comment'. The X and Y values are positions with respect to an arbitrary origin. All numeric inputs are in meters.

BASLIN: The baselines between transponders are calculated using the X-Y positions and are written into the list file along with the X-Y positions.

XPMAIN

Main Program	Subroutines Level 1	Level 2
XPMAIN		
	XPCMLD	
	XPINPT	
	XXCOR	
LOOP:	XPREAD	XXCOR
	XPLOOP	XPFIL (perturbs *FLIP* position)
		XPFIL (perturbs transponder positions)
	XPURGE	
	BASLN2	
	TMDATE	
END LOOP:		
	BASLN2	
	BASLN2	
	XPRINT	
	TMDATE	
	XPRIN2	
	EXIT	

XPMAIN perturbs the xy positions of *FLIP* and each transponder until the RMS error is minimized. There are four files involved:

XPCOM.DAT: defines a common area which is set up by XPLOAD. The common area contains the transponder information, nominal sound speed, plotter parameters, and logical device definitions. Sound speed and plotter parameters are obsolete.

xxx.dat: contains the time, X and Y position and the depth of the interrogation hydrophone which is deployed from *FLIP* and slant range information for each transponder for a series of fixes. This file is set up manually by reading the slant range for each transponder off of the chart recorder output (if surface ship is used for the survey, by also recording the corresponding LORAN-C or SATNAV position and converting to meters from an arbitrary origin). The same xy *FLIP* position is used for all fixes due to the limited range of the survey.

xxx.trsout is an output file containing the adjusted transponder positions.

xxx.pos is an output file containing the adjusted *FLIP* positions.

XPCMLD: is a subroutine which reads the data in the transponder common area from the file called *XPCOM.DAT* which is written by XPLOAD.

XPINPT: opens the data file *xxx.dat*, the transponder file *xxx.trsout* and inquires as to the minimum number of ranges acceptable (default=3). The option of selecting specific transponders to be removed from the net is available but will not be used in

this demonstration as the minimum number of transponders is 3.

XPMAIN requests an operator input for the RMS error factor for fix rejection (default=2) and saves the initial transponder positions for later comparison.

LOOP:

XPREAD: opens the data file *xxx.dat* and calculates the horizontal range (XY projection) from the source hydrophone on *FLIP* to each transponder and for each fix using the source depth, transponder depth and slant range.

$$HH = CRANS(ntr, npos) = \sqrt{S^2 - D^2}$$

where ntr indicates a particular transponder, npos indicates a particular fix or position, S is the slant range from the source to that transponder and D is the transponder depth - source depth.

XPLOOP: initializes the minimization of the RMS error in the horizontal range. This is accomplished in a loop in which XPLOOP calls XPFIL transfering the appropriate parameters to first adjust the *FLIP* positions (npos fixes) for each transponder and then adjust the transponder positions for each *FLIP* position (fix). The RMS error is the error after perturbing the transponder positions. If the reduction in normalized RMS error from one loop to the next is less than 0.015 or is the number of loops exceeds 30, XPLOOP is terminated.

XPFIL: adjusts the xy positions. The first sequence during which it is called within the XPLOOP loop, it sums the squared error in the xy projections, $\sum(rngec - HH)^2$ where rngec is the xy projection calculated from the xy positions of the *FLIP* and the transponder and HH is calculated as described in XPREAD, for each transponder. The *FLIP* position is then adjusted: $X = X + DX * RATIO * GAIN / N$ where X is the original X position of the *FLIP*, DX = the difference in X-positions of the *FLIP* and the transponder, RATIO = (HH-RNGEC)/RNGEC, GAIN = 1.5, N = number of transponders. If the root-mean of the squared error is less than 0.25 or if the normalized reduction in rms error is less than 0.035 it returns to XPLOOP. Otherwise the squared error is cleared and calculated again. This continues until the rms error satisfies one of the above criteria or the number of adjustments exceeds 30. Although the final error calculated from the *FLIP* position adjustment .s passed back to XPLOOP it is not used. The parameters associated with the next *FLIP* position are amassed and XPFIL is called again. This continues for all *FLIP* positions or fixes in *xxx.dat*. The second sequence during which XPFIL is called the proceedure is essentially identical except the the squared error is summed for each *FLIP* position, the position adjusted is the transponder position, and N is the number of *FLIP* positions.

XPMAIN then evaluates the rms error passed back by XPLOOP for the following criteria:

If the RMS error * $\sqrt{\text{number of fixes in xxx.dat}}$ is < 1.0.

If the normalized reduction in RMS error per loop [(OLD error - current error) / OLD error] < 0.035. This terminates the loop if the reduction in error is less than a specified value or if the error increases.

If the RMS error is less than a specified number (0.75).

If any of the above criteria are met, XPMAIN jumps out of the LOOP and begins the print sequence. The error must converge to one of these criteria, there is no limit as to the number of times the LOOP may be

executed.

XPURGE removes any *FLIP* fixes which have an rms error greater than the fix rejection parameter input from the keyboard earlier (default=2) times the total transponder rms error.

BASLN2 calculates baselines from transponder xy positions and writes to the screen.

XPMAIN stores the new transponder positions.

TMDATE calculate real time and date from a system call.

XPMAIN writes the date, time and transponder positions to an intermediate file which is defined currently as the screen.

GOTO LOOP

BASLN2 calculates baselines from initial transponder xy positions and writes to the screen.

BASLN2 calculates baselines from final transponder xy positions and writes to the screen.

XPMAIN calculates the difference between the initial transponder baselines and the final transponder baselines and prints to the screen.

XPRINT writes to the screen the adjusted transponder ranges and the time, X, Y, error and depth for the *FLIP* transceiver.

XPRIN2 writes to the output file *xxx.pos* the time, X, Y, error and depth for the *FLIP* source, a quality factor, and the original slant ranges to the transponders in the same format as the input file *xxx.dat*.

```
Dec 21 21:09 1988  AppA.11.doc Page 1


-----------------------------------------------------------------

SAMPLE RUN of xpload

N51>xpload
  LUTRC =   19
  XPCOM.DAT FOUND, OLD FILE OPENED...
  LFINL,LINTER,LERR,LSPAR=  6  6  6  8
  LSCRN,LNEW=  6  37
   LSPAR =  8
Listing file : xxx.lst
Transponder file : TRS.trs
Open new file ?(y/n):y
Descriptive title :Display model
Number of Transponders : 3
 Input transponder  1...

Label,X,Y,Depth,Comment : Red, 591., 4158., 4564., transponder 1
 Input transponder  2...

Label,X,Y,Depth,Comment : Grn, 2600., 797., 4566., transponder 2
 Input transponder  3...

Label,X,Y,Depth,Comment : Blu, 4428., 4771., 4573., transponder 3

 Display model
  #  LABEL       X       Y     DEPTH    COMMENTS

  1  Red      591.   4158.   4564.  transponder 1
  2  Grn     2600.    797.   4566.  transponder 2
  3  Blu     4428.   4771.   4573.  transponder 3

Positions ok ?(y/n):y
 SURFV =  1500.0 DEEPV =   1500.0 ...Velocities OK ?(y/n):y
  LUTRC =   19
  XPCOM.DAT FOUND, OLD FILE OPENED...
  LFINL,LINTER,LERR,LSPAR=  6  6  6  8
  LSCRN,LNEW=  6  37

-----------------------------------------------------------------

OUTPUT FILE:  xxx.lst

 Transponder data file... TRS.trs

 Display model
  #  LABEL       X       Y     DEPTH    COMMENTS

  1  Red      591.   4158.   4564.  transponder 1
  2  Grn     2600.    797.   4566.  transponder 2
  3  Blu     4428.   4771.   4573.  transponder 3

          Red      Grn
 Grn   3915.7
```

```
Dec 21 21:09 1988  AppA.11.doc Page 2

 Blu   3885.7  4374.3

-----------------------------------------------------------------

OUTPUT FILE:  TRS.trs

Display model
 Red          591.   4158.  4564. transponder 1
 Grn         2600.    797.  4566. transponder 2
 Blu         4428.   4771.  4573. transponder 3
```

```
Dec 21 20:56 1988  AppA.22.doc Page 1

-----------------------------------------------------------------
XPMAIN SAMPLE RUN:

N51)xpmain
  LUTRC =   19
  XPCOM.DAT FOUND, OLD FILE OPENED...
  LFINL,LINTER,LERR,LSPAR=  6  6  6  8
  LSCRN,LNEW=  6  37
Navigation data file : flip.indat
Looped output transponder file : Trs.out
Minimum no. of ranges acceptable ?
 Re Gr Bl                transponders are currently in the system.
 If you wish to adjust all positions,just press "return".
 If you wish to select particular transponders for
 looping, type in their names one at a time followed
 by "return". To terminate the list of names,
 press "return" without entering a name.


Transponder name :
RMS error factor for fix rejection (default=2) :
Do you want to correct the slant ranges for sound     speed? (NO=0,YES=1) : 1
 Navigation file is flip.indat
             1  2400.  3200.  0.  89.   0. 4920. 5130. 5108.
             2  2400.  3200.  0.  89.   0. 4906. 5086. 5087.
             3  2400.  3200.  0.  89.   0. 4907. 5090. 5085.
             4  2400.  3200.  0.  89.   0. 4914. 5074. 5091.
             5  2400.  3200.  0.  89.   0. 4917. 5066. 5093.

             ............  displays navigation data file  ........

           420  2400.  3200.  0.  89.   0. 4896. 5018. 5173.
           421  2400.  3200.  0.  89.   0. 4899. 5024. 5163.
           422  2400.  3200.  0.  89.   0. 4897. 5035. 5151.
           423  2400.  3200.  0.  89.   0. 4902. 5040. 5140.
           424  2400.  3200.  0.  89.   0. 4898. 5036. 5150.


   424 POSITIONS IN STORE
 LOOP NO     1 HAS AN ERROR OF  9.1052
 LOOP NO     2 HAS AN ERROR OF  3.0377
 LOOP NO     3 HAS AN ERROR OF  2.9809
 LOOP NO     4 HAS AN ERROR OF  2.9773
 LOOP NO     4 HAS AN ERROR OF  2.9773


*** POSITION(S) REJECTED...***
    1.0   2462.5   3315.4  56.839  2044.65  2504.65  2446.59
 Baseline lengths derived by looping...

       Re          Gr
 Gr  3793.86572
 Bl  3786.06030  4258.33643
```

```
Dec 21 20:56 1988  AppA.22.doc Page 2

 17-Dec-88 13:46:28 looping, RMS =   2.98 based on  423 fixes from flip.indat
 Red        646.   4128.  4564.        Re      Gr
 Grn       2593.    872.  4566.      3793.9
 Blu       4383.   4736.  4573.      3786.1  4258.3


   423 POSITIONS IN STORE
 LOOP NO     1 HAS AN ERROR OF  1.1159
 LOOP NO     2 HAS AN ERROR OF  1.1138
 LOOP NO     2 HAS AN ERROR OF  1.1138


*** POSITION(S) REJECTED...***
     3.0    2473.5    3288.2    2.364  2013.17  2421.68  2398.19
     7.0    2455.2    3228.2    2.333  2018.04  2357.96  2444.50
     8.0    2471.7    3233.8    2.638  2030.17  2362.25  2427.73
    10.0    2464.8    3242.2    4.950  2018.04  2368.67  2425.64
    24.0    2437.2    3186.6    2.866  2020.47  2316.91  2483.95
    27.0    2457.2    3232.6    2.243  2018.04  2362.25  2440.32
    32.0    2446.0    3235.1    3.276  2005.85  2364.39  2446.59
    51.0    2409.2    3253.3    3.494  1971.37  2392.10  2471.54
    56.0    2435.4    3247.8    2.323  1996.05  2383.61  2452.84
    60.0    2417.0    3272.1    2.278  1968.89  2409.03  2452.84
    74.0    2411.7    3268.0    2.343  1961.43  2400.58  2454.92
    75.0    2417.2    3266.8    2.837  1966.40  2398.46  2450.76
   199.0    2358.0    3180.5    2.646  1958.94  2323.43  2555.40
   308.0    2437.0    3191.8    2.414  2022.90  2327.77  2486.01
   352.0    2400.5    3100.8    2.340  2035.01  2239.74  2571.52
   353.0    2394.0    3070.9    2.301  2044.65  2210.53  2595.56
 Baseline lengths derived by looping...

       Re          Gr
 Gr  3793.67017
 Bl  3785.85913  4258.11768


 17-Dec-88 13:46:32 looping, RMS =   1.11 based on  407 fixes from flip.indat
 Red        646.   4128.  4564.        Re      Gr
 Grn       2593.    872.  4566.      3793.7
 Blu       4383.   4736.  4573.      3785.9  4258.1


   407 POSITIONS IN STORE
 LOOP NO     1 HAS AN ERROR OF 0.98780
 LOOP NO     2 HAS AN ERROR OF 0.98772
 LOOP NO     2 HAS AN ERROR OF 0.98772


*** POSITION(S) REJECTED...***
    11.0    2470.2    3226.6    2.069  2032.59  2355.82  2434.03
    16.0    2460.4    3256.1    2.165  2010.73  2385.73  2423.53
    76.0    2418.3    3259.7    2.092  1971.37  2392.10  2454.92
    90.0    2365.6    3117.9    2.033  1996.05  2259.78  2587.56
    98.0    2364.0    3088.5    1.991  2005.85  2226.29  2603.53
    99.0    2371.1    3114.2    2.263  1998.50  2250.89  2581.56
   101.0    2366.2    3073.0    2.146  2015.60  2210.53  2611.49
```

```
 180.0   2379.0   3152.7   2.237  1986.21  2288.49  2551.36
 220.0   2352.1   3174.0   1.994  1956.44  2316.91  2563.47
 231.0   2378.5   3196.2   1.999  1968.89  2336.43  2529.04
 246.0   2364.7   3170.9   1.978  1968.89  2312.56  2555.40
 328.0   2422.6   3134.1   2.023  2037.42  2270.86  2533.11
 355.0   2379.0   3023.8   2.104  2056.65  2164.97  2637.23
 378.0   2371.7   3100.8   2.243  2005.85  2237.50  2589.57
 384.0   2373.2   3064.5   2.110  2030.17  2206.01  2615.46
 399.0   2375.6   3206.8   2.037  1961.43  2347.21  2524.97
 404.0   2384.7   3218.4   1.986  1963.92  2357.96  2510.67
Baseline lengths derived by looping...

        Re           Gr
Gr  3793.64209
Bl  3785.82178  4258.07617


17-Dec-88 13:46:34 looping, RMS =   0.99 based on  390 fixes from flip.indat
Red          646.   4128.  4564.          Re       Gr
Grn         2593.    872.  4566.        3793.6
Blu         4383.   4736.  4573.        3785.8  4258.1


  390 POSITIONS IN STORE
LOOP NO     1 HAS AN ERROR OF 0.90996
LOOP NO     2 HAS AN ERROR OF 0.90995
LOOP NO     2 HAS AN ERROR OF 0.90995


*** POSITION(S) REJECTED...***
   15.0   2441.2   3250.5   1.982  1996.05  2381.47  2442.40
   26.0   2447.0   3229.3   1.899  2010.73  2360.11  2450.76
   92.0   2331.6   3080.6   1.912  1986.21  2226.29  2637.23
  146.0   2381.1   3159.5   1.824  1988.67  2299.46  2549.34
  174.0   2369.4   3184.6   1.917  1966.40  2325.60  2543.26
  179.0   2377.2   3165.3   1.890  1978.80  2301.64  2545.29
  205.0   2362.0   3168.3   1.979  1963.92  2306.01  2555.40
  243.0   2370.8   3156.3   1.846  1981.27  2297.26  2559.44
  350.0   2374.4   3094.2   1.949  2015.60  2235.26  2595.56
  372.0   2356.7   2989.1   1.864  2056.65  2132.62  2676.47
Baseline lengths derived by looping...

        Re           Gr
Gr  3793.63672
Bl  3785.80737  4258.05957


17-Dec-88 13:46:37 looping, RMS =   0.91 based on  380 fixes from flip.indat
Red          646.   4128.  4564.          Re       Gr
Grn         2593.    872.  4566.        3793.6
Blu         4383.   4736.  4573.        3785.8  4258.1


  380 POSITIONS IN STORE
LOOP NO     1 HAS AN ERROR OF 0.86842
LOOP NO     2 HAS AN ERROR OF 0.86840
```



```
LOOP NO     2 HAS AN ERROR OF 0.86840


*** POSITION(S) REJECTED...***
  117.0   2385.9   3070.6   1.789  2037.42  2210.53  2601.54
  155.0   2383.8   3157.0   1.799  1988.67  2292.88  2545.29
  156.0   2383.8   3157.0   1.799  1988.67  2292.88  2545.29
  245.0   2362.0   3170.8   1.808  1966.40  2312.56  2557.42
  351.0   2392.8   3125.2   1.738  2015.60  2264.22  2561.45
  402.0   2380.3   3220.4   1.775  1958.94  2360.11  2512.72
Baseline lengths derived by looping...

        Re           Gr
Gr  3793.62378
Bl  3785.79468  4258.04053


17-Dec-88 13:46:40 looping, RMS =   0.87 based on  374 fixes from flip.indat
Red          646.   4128.  4564.          Re       Gr
Grn         2593.    872.  4566.        3793.6
Blu         4383.   4736.  4573.        3785.8  4258.0


  374 POSITIONS IN STORE
LOOP NO     1 HAS AN ERROR OF 0.84567
LOOP NO     2 HAS AN ERROR OF 0.84565
LOOP NO     2 HAS AN ERROR OF 0.84565


*** POSITION(S) REJECTED...***
    5.0   2482.7   3241.8   1.720  2037.42  2370.81  2415.11
   97.0   2339.3   3062.8   1.721  1998.50  2203.74  2639.20
  147.0   2381.7   3155.3   1.722  1991.13  2295.07  2551.36
  237.0   2391.6   3131.1   1.724  2008.29  2266.43  2555.40
  251.0   2374.3   3120.6   1.722  1998.50  2257.56  2575.54
  331.0   2396.2   3083.0   1.701  2039.84  2221.80  2585.56
Baseline lengths derived by looping...

        Re           Gr
Gr  3793.60962
Bl  3785.78149  4258.02051


17-Dec-88 13:46:42 looping, RMS =   0.85 based on  368 fixes from flip.indat
Red          646.   4128.  4564.          Re       Gr
Grn         2593.    872.  4566.        3793.6
Blu         4383.   4736.  4573.        3785.8  4258.0


  368 POSITIONS IN STORE
LOOP NO     1 HAS AN ERROR OF 0.82367
LOOP NO     2 HAS AN ERROR OF 0.82367
LOOP NO     2 HAS AN ERROR OF 0.82367


*** POSITION(S) REJECTED...***
```

```
  12.0    2465.7    3176.1    1.698  2051.86  2306.01  2469.47
  73.0    2410.7    3239.5    1.705  1973.85  2372.95  2473.61
  83.0    2397.8    3119.1    1.664  2022.90  2257.56  2561.45
 149.0    2381.6    3163.3    1.708  1983.74  2299.46  2543.26
 334.0    2403.2    3107.5    1.657  2030.17  2241.97  2561.45
Baseline lengths derived by looping...

       Re          Gr
Gr  3793.62036
Bl  3785.79077  4258.02490


17-Dec-88 13:46:45 looping, RMS =   0.82 based on  363 fixes from flip.indat
Red          646.   4128.  4564.         Re      Gr
Grn         2593.    872.  4566.       3793.6
Blu         4383.   4736.  4573.       3785.8  4258.0


  363 POSITIONS IN STORE
LOOP NO      1 HAS AN ERROR OF 0.80533
LOOP NO      2 HAS AN ERROR OF 0.80530
LOOP NO      2 HAS AN ERROR OF 0.80530

*** POSITION(S) REJECTED...***
   9.0    2453.8    3238.1    1.659  2013.17  2368.67  2440.32
  28.0    2460.0    3223.4    1.619  2025.33  2353.67  2444.50
  43.0    2422.4    3226.2    1.619  1993.59  2362.25  2475.68
 100.0    2368.3    3068.1    1.647  2020.47  2206.01  2613.48
 151.0    2389.8    3153.0    1.657  1996.05  2288.49  2543.26
 152.0    2391.4    3150.9    1.652  1998.50  2286.29  2543.26
 153.0    2389.8    3153.0    1.657  1996.05  2288.49  2543.26
 204.0    2367.3    3168.5    1.652  1968.89  2306.01  2551.36
 386.0    2391.8    3093.0    1.620  2027.75  2228.54  2579.55
Baseline lengths derived by looping...

       Re          Gr
Gr  3793.64282
Bl  3785.81177  4258.04297


17-Dec-88 13:46:47 looping, RMS =   0.81 based on  354 fixes from flip.indat
Red          646.   4128.  4564.         Re      Gr
Grn         2593.    872.  4566.       3793.6
Blu         4383.   4736.  4573.       3785.8  4258.0


  354 POSITIONS IN STORE
LOOP NO      1 HAS AN ERROR OF 0.77212
LOOP NO      2 HAS AN ERROR OF 0.77196
LOOP NO      2 HAS AN ERROR OF 0.77196

*** POSITION(S) REJECTED...***
  53.0    2432.7    3262.2    1.610  1983.74  2394.23  2442.40
 124.0    2359.0    3114.4    1.596  1988.67  2253.12  2591.56
```



```
 181.0    2370.6    3164.3    1.631  1973.85  2301.64  2551.36
 226.0    2375.8    3138.3    1.605  1991.13  2275.28  2563.47
 230.0    2365.1    3174.8    1.638  1963.92  2312.56  2549.34
 253.0    2364.8    3160.3    1.565  1973.85  2301.64  2561.45
 310.0    2408.1    3202.5    1.589  1988.67  2336.43  2498.36
 311.0    2410.4    3196.2    1.613  1993.59  2329.94  2500.42
 401.0    2380.5    3209.8    1.564  1963.92  2349.37  2518.85
Baseline lengths derived by looping...

       Re          Gr
Gr  3793.69336
Bl  3785.85645  4258.08936


17-Dec-88 13:46:50 looping, RMS =   0.77 based on  345 fixes from flip.indat
Red          646.   4128.  4564.         Re      Gr
Grn         2593.    872.  4566.       3793.7
Blu         4383.   4736.  4573.       3785.9  4258.1


  345 POSITIONS IN STORE
LOOP NO      1 HAS AN ERROR OF 0.73827
LOOP NO      2 HAS AN ERROR OF 0.73813
LOOP NO      2 HAS AN ERROR OF 0.73813
Initial transponder baseline lengths...

       Re          Gr
Gr  3915.66113
Bl  3885.65796  4374.27246


Baseline lengths derived by looping...

       Re          Gr
Gr  3793.73779
Bl  3785.89697  4258.12940


Differences between initial and "looped" baselines...

         Re             Gr
Gr     -121.92
Bl      -99.761        -116.14
   TRX      TRY      TRZ       ID
  646.0   4127.6   4564.0      Re
 2593.2    871.7   4566.0      Gr
 4382.7   4735.6   4573.2      Bl



      TIME       X         Y       ERROR


  1     2.0    2470.1    3281.8    0.010  2010.73  2413.25  2402.43
  2     4.0    2479.4    3256.5    0.352  2030.17  2387.86  2410.89
  3     6.0    2502.3    3219.6    0.306  2066.21  2349.37  2415.11
```

```
  4   13.0   2464.6   3167.4   0.130  2056.65  2299.46  2477.75
  5   14.0   2436.7   3164.4   0.757  2032.59  2297.26  2500.42

   .................. displays position output file ..........

 340  419.0  2362.1   3102.4   0.603  1998.50  2241.97  2597.56
 341  420.0  2363.8   3128.0   1.355  1986.21  2266.43  2579.55
 342  421.0  2379.4   3141.9   0.524  1993.59  2279.69  2559.44
 343  422.0  2389.0   3167.8   1.235  1988.67  2303.83  2535.14
 344  423.0  2408.9   3179.8   0.636  2000.95  2314.73  2512.72
 345  424.0  2391.6   3169.2   0.314  1991.13  2306.01  2533.11
New position output: xxx.pos
 XPRINT
26.3u 1.6s 1:42 27% 40+56k 11+14io 10pf+0w
N52>exit
N53>
script done on Sat Dec 17 13:47:22 1988

-------------------------------------------------------

FLIP.INDAT:

.sp
254/ 219     1  2400.  3200.  0  89  0  4920  5130  5108
254/1700     2  2400.  3200.  0  89  0  4906  5086  5087
254/1856     3  2400.  3200.  0  89  0  4907  5090  5085
254/2058     4  2400.  3200.  0  89  0  4914  5074  5091
254/2306     5  2400.  3200.  0  89  0  4917  5066  5093
255/ 100     6  2400.  3200.  0  89  0  4929  5056  5093
255/ 306     7  2400.  3200.  0  89  0  4909  5060  5107
255/ 500     8  2400.  3200.  0  89  0  4914  5062  5099
255/ 700     9  2400.  3200.  0  89  0  4907  5065  5105
255/ 900    10  2400.  3200.  0  89  0  4909  5065  5098
255/1008    11  2400.  3200.  0  89  0  4915  5059  5102
255/1100    12  2400.  3200.  0  89  0  4923  5036  5119
255/1300    13  2400.  3200.  0  89  0  4925  5033  5123
255/1450    14  2400.  3200.  0  89  0  4915  5032  5134
255/1614    15  2400.  3200.  0  89  0  4900  5071  5106
255/1712    16  2400.  3200.  0  89  0  4906  5073  5097
255/1903    17  2400.  3200.  0  89  0  4904  5075  5100
255/2013    18  2400.  3200.  0  89  0  4909  5066  5103
255/2100    19  2400.  3200.  0  89  0  4907  5066  5106
255/2300    20  2400.  3200.  0  89  0  4922  5037  5121
256/   4    21  2400.  3200.  0  89  0  4918  5041  5121
256/ 100    22  2400.  3200.  0  89  0  4923  5038  5118
256/ 200    23  2400.  3200.  0  89  0  4913  5042  5127
256/ 300    24  2400.  3200.  0  89  0  4910  5041  5126
256/ 400    25  2400.  3200.  0  89  0  4910  5049  5120
256/ 516    26  2400.  3200.  0  89  0  4906  5061  5110
256/ 610    27  2400.  3200.  0  89  0  4909  5062  5105
256/ 708    28  2400.  3200.  0  89  0  4912  5058  5107
256/ 800    29  2400.  3200.  0  89  0  4912  5054  5112
256/ 905    30  2400.  3200.  0  89  0  4912  5052  5117
256/ 958    31  2400.  3200.  0  89  0  4904  5060  5115
256/1055    32  2400.  3200.  0  89  0  4904  5063  5108
256/1203    33  2400.  3200.  0  89  0  4906  5059  5115
```



```
256/1300    34  2400.  3200.  0  89  0  4904  5063  5112
256/1400    35  2400.  3200.  0  89  0  4900  5067  5115
256/1500    36  2400.  3200.  0  89  0  4897  5070  5115
256/1627    37  2400.  3200.  0  89  0  4887  5085  5110
256/1658    38  2400.  3200.  0  89  0  4892  5079  5111
256/1834    39  2400.  3200.  0  89  0  4893  5073  5116
256/1855    40  2400.  3200.  0  89  0  4898  5064  5120
256/1903    41  2400.  3200.  0  89  0  4900  5060  5122
256/1959    42  2400.  3200.  0  89  0  4899  5064  5119
256/2100    43  2400.  3200.  0  89  0  4899  5062  5122
256/2210    44  2400.  3200.  0  89  0  4897  5070  5115
256/2300    45  2400.  3200.  0  89  0  4895  5074  5113
257/   0    46  2400.  3200.  0  89  0  4892  5078  5111
257/ 100    47  2400.  3200.  0  89  0  4897  5073  5111
257/ 200    48  2400.  3200.  0  89  0  4907  5056  5117
257/ 300    49  2400.  3200.  0  89  0  4897  5062  5123
257/ 405    50  2400.  3200.  0  89  0  4893  5065  5122
257/ 500    51  2400.  3200.  0  89  0  4890  5076  5120
257/ 600    52  2400.  3200.  0  89  0  4888  5086  5106
257/ 700    53  2400.  3200.  0  89  0  4895  5077  5106
257/ 800    54  2400.  3200.  0  89  0  4893  5076  5111
257/ 900    55  2400.  3200.  0  89  0  4896  5076  5107
257/ 957    56  2400.  3200.  0  89  0  4900  5072  5111
257/1055    57  2400.  3200.  0  89  0  4897  5071  5111
257/1200    58  2400.  3200.  0  89  0  4901  5068  5109
257/1300    59  2400.  3200.  0  89  0  4894  5076  5110
257/1400    60  2400.  3200.  0  89  0  4889  5084  5111
257/1500    61  2400.  3200.  0  89  0  4894  5072  5114
257/1600    62  2400.  3200.  0  89  0  4887  5077  5117
257/1700    63  2400.  3200.  0  89  0  4881  5094  5107
257/1800    64  2400.  3200.  0  89  0  4884  5087  5110
257/1900    65  2400.  3200.  0  89  0  4887  5083  5111
257/2007    66  2400.  3200.  0  89  0  4889  5084  5107
257/2200    67  2400.  3200.  0  89  0  4888  5092  5101
258/   0    68  2400.  3200.  0  89  0  4897  5081  5101
258/ 100    69  2400.  3200.  0  89  0  4913  5053  5115
258/ 200    70  2400.  3200.  0  89  0  4908  5050  5123
258/ 300    71  2400.  3200.  0  89  0  4904  5054  5122
258/ 400    72  2400.  3200.  0  89  0  4897  5058  5127
258/ 500    73  2400.  3200.  0  89  0  4891  5067  5121
258/ 600    74  2400.  3200.  0  89  0  4886  5080  5112
258/ 700    75  2400.  3200.  0  89  0  4888  5079  5110
258/ 810    76  2400.  3200.  0  89  0  4890  5076  5112
258/ 901    77  2400.  3200.  0  89  0  4897  5064  5119
258/ 955    78  2400.  3200.  0  89  0  4901  5050  5129
258/1055    79  2400.  3200.  0  89  0  4898  5057  5125
258/1200    80  2400.  3200.  0  89  0  4899  5050  5132
258/1300    81  2400.  3200.  0  89  0  4910  5037  5135
258/1400    82  2400.  3200.  0  89  0  4911  5020  5153
258/1500    83  2400.  3200.  0  89  0  4911  5014  5164
258/1600    84  2400.  3200.  0  89  0  4893  5015  5184
258/1700    85  2400.  3200.  0  89  0  4887  5019  5186
258/1800    86  2400.  3200.  0  89  0  4886  5026  5179
258/1900    87  2400.  3200.  0  89  0  4879  5036  5174
258/1958    88  2400.  3200.  0  89  0  4890  5029  5170
258/2057    89  2400.  3200.  0  89  0  4895  5021  5175
```

258/2157	90	2400.	3200.	0	89	0	4900	5015	5177
258/2255	91	2400.	3200.	0	89	0	4895	5008	5190
259/ 0	92	2400.	3200.	0	89	0	4896	5000	5202
259/ 100	93	2400.	3200.	0	89	0	4898	5016	5175
259/ 200	94	2400.	3200.	0	89	0	4899	5020	5170
259/ 305	95	2400.	3200.	0	89	0	4906	5000	5188
259/ 408	96	2400.	3200.	0	89	0	4907	4981	5212
259/ 500	97	2400.	3200.	0	89	0	4901	4990	5203
259/ 600	98	2400.	3200.	0	89	0	4904	5000	5185
259/ 700	99	2400.	3200.	0	89	0	4901	5011	5174
259/ 801	100	2400.	3200.	0	89	0	4910	4991	5190
259/ 859	101	2400.	3200.	0	89	0	4908	4993	5189
259/ 957	102	2400.	3200.	0	89	0	4913	4988	5192
259/1100	103	2400.	3200.	0	89	0	4910	4992	5192
259/1200	104	2400.	3200.	0	89	0	4910	4991	5194
259/1300	105	2400.	3200.	0	89	0	4908	4995	5188
259/1400	106	2400.	3200.	0	89	0	4906	5000	5187
259/1500	107	2400.	3200.	0	89	0	4900	5004	5187
259/1600	108	2400.	3200.	0	89	0	4895	5019	5176
259/1700	109	2400.	3200.	0	89	0	4896	5018	5173
259/1800	110	2400.	3200.	0	89	0	4904	5007	5179
259/1900	111	2400.	3200.	0	89	0	4903	5007	5180
259/2011	112	2400.	3200.	0	89	0	4907	5009	5171
259/2100	113	2400.	3200.	0	89	0	4912	5010	5166
259/2200	114	2400.	3200.	0	89	0	4909	5009	5171
259/2300	115	2400.	3200.	0	89	0	4918	4993	5181
260/ 0	116	2400.	3200.	0	89	0	4916	4998	5177
260/ 100	117	2400.	3200.	0	89	0	4917	4993	5184
260/ 200	118	2400.	3200.	0	89	0	4916	4991	5187
260/ 300	119	2400.	3200.	0	89	0	4915	4990	5187
260/ 400	120	2400.	3200.	0	89	0	4903	4992	5203
260/ 500	121	2400.	3200.	0	89	0	4893	4997	5208
260/ 600	122	2400.	3200.	0	89	0	4892	5008	5192
260/ 700	123	2400.	3200.	0	89	0	4896	5010	5183
260/ 800	124	2400.	3200.	0	89	0	4897	5012	5179
260/ 900	125	2400.	3200.	0	89	0	4899	5012	5177
260/1000	126	2400.	3200.	0	89	0	4902	5007	5181
260/1100	127	2400.	3200.	0	89	0	4897	5018	5172
260/1200	128	2400.	3200.	0	89	0	4895	5031	5159
260/1300	129	2400.	3200.	0	89	0	4900	5031	5153
260/1400	130	2400.	3200.	0	89	0	4904	5024	5157
260/1500	131	2400.	3200.	0	89	0	4906	5022	5159
260/1600	132	2400.	3200.	0	89	0	4904	5024	5159
260/1700	133	2400.	3200.	0	89	0	4904	5009	5175
260/1800	134	2400.	3200.	0	89	0	4900	5019	5170
260/1900	135	2400.	3200.	0	89	0	4897	5021	5170
260/2000	136	2400.	3200.	0	89	0	4897	5027	5163
260/2100	137	2400.	3200.	0	89	0	4897	5027	5163
260/2200	138	2400.	3200.	0	89	0	4898	5029	5159
260/2301	139	2400.	3200.	0	89	0	4897	5022	5169
260/2356	140	2400.	3200.	0	89	0	4897	5024	5166
261/ 6	141	2400.	3200.	0	89	0	4896	5026	5165
261/ 100	142	2400.	3200.	0	89	0	4895	5034	5158
261/ 200	143	2400.	3200.	0	89	0	4897	5027	5162
261/ 334	144	2400.	3200.	0	89	0	4893	5033	5161
261/ 347	145	2400.	3200.	0	89	0	4894	5033	5160

261/ 358	146	2400.	3200.	0	89	0	4897	5033	5158
261/ 458	147	2400.	3200.	0	89	0	4898	5031	5159
261/ 554	148	2400.	3200.	0	89	0	4896	5030	5159
261/ 659	149	2400.	3200.	0	89	0	4895	5033	5155
261/ 800	150	2400.	3200.	0	89	0	4896	5034	5153
261/ 902	151	2400.	3200.	0	89	0	4900	5028	5155
261/1001	152	2400.	3200.	0	89	0	4901	5027	5155
261/1101	153	2400.	3200.	0	89	0	4900	5028	5155
261/1120	154	2400.	3200.	0	89	0	4899	5028	5157
261/1144	155	2400.	3200.	0	89	0	4897	5030	5156
261/1204	156	2400.	3200.	0	89	0	4897	5030	5156
261/1251	157	2400.	3200.	0	89	0	4895	5031	5158
261/1328	158	2400.	3200.	0	89	0	4891	5038	5157
261/1355	159	2400.	3200.	0	89	0	4888	5046	5152
261/1450	160	2400.	3200.	0	89	0	4887	5051	5146
261/1558	161	2400.	3200.	0	89	0	4888	5048	5150
261/1702	162	2400.	3200.	0	89	0	4885	5050	5150
261/1757	163	2400.	3200.	0	89	0	4885	5050	5152
261/1832	164	2400.	3200.	0	89	0	4888	5044	5153
261/1859	165	2400.	3200.	0	89	0	4888	5046	5151
261/2010	166	2400.	3200.	0	89	0	4888	5047	5149
261/2103	167	2400.	3200.	0	89	0	4892	5042	5151
261/2203	168	2400.	3200.	0	89	0	4895	5037	5154
261/2259	169	2400.	3200.	0	89	0	4891	5035	5160
261/2358	170	2400.	3200.	0	89	0	4885	5044	5157
262/ 101	171	2400.	3200.	0	89	0	4888	5042	5156
262/ 159	172	2400.	3200.	0	89	0	4888	5046	5152
262/ 302	173	2400.	3200.	0	89	0	4885	5052	5150
262/ 400	174	2400.	3200.	0	89	0	4888	5045	5155
262/ 501	175	2400.	3200.	0	89	0	4886	5047	5153
262/ 559	176	2400.	3200.	0	89	0	4890	5041	5154
262/ 700	177	2400.	3200.	0	89	0	4890	5039	5156
262/ 802	178	2400.	3200.	0	89	0	4892	5036	5156
262/ 859	179	2400.	3200.	0	89	0	4893	5034	5156
262/1004	180	2400.	3200.	0	89	0	4896	5028	5159
262/1104	181	2400.	3200.	0	89	0	4891	5034	5159
262/1159	182	2400.	3200.	0	89	0	4887	5041	5156
262/1256	183	2400.	3200.	0	89	0	4887	5044	5153
262/1403	184	2400.	3200.	0	89	0	4890	5043	5150
262/1502	185	2400.	3200.	0	89	0	4887	5045	5152
262/1600	186	2400.	3200.	0	89	0	4887	5049	5149
262/1707	187	2400.	3200.	0	89	0	4885	5051	5150
262/1800	188	2400.	3200.	0	89	0	4883	5054	5150
262/1853	189	2400.	3200.	0	89	0	4882	5055	5148
262/1901	190	2400.	3200.	0	89	0	4886	5048	5152
262/2104	191	2400.	3200.	0	89	0	4887	5046	5152
262/2156	192	2400.	3200.	0	89	0	4892	5038	5156
262/2254	193	2400.	3200.	0	89	0	4893	5032	5160
263/ 4	194	2400.	3200.	0	89	0	4895	5028	5164
263/ 57	195	2400.	3200.	0	89	0	4894	5024	5171
263/ 219	196	2400.	3200.	0	89	0	4887	5036	5162
263/ 303	197	2400.	3200.	0	89	0	4887	5036	5166
263/ 401	198	2400.	3200.	0	89	0	4882	5042	5164
263/ 459	199	2400.	3200.	0	89	0	4885	5044	5161
263/ 600	200	2400.	3200.	0	89	0	4886	5040	5161
263/ 703	201	2400.	3200.	0	89	0	4886	5035	5168

Dec 21 20:56 1988 AppA.22.doc Page 11

263/ 801	202	2400.	3200.	0	89	0	4887	5034	5166
263/ 902	203	2400.	3200.	0	89	0	4890	5036	5159
263/1000	204	2400.	3200.	0	89	0	4889	5036	5159
263/1122	205	2400.	3200.	0	89	0	4887	5036	5161
263/1209	206	2400.	3200.	0	89	0	4890	5034	5161
263/1302	207	2400.	3200.	0	89	0	4889	5037	5160
263/1412	208	2400.	3200.	0	89	0	4888	5039	5160
263/1504	209	2400.	3200.	0	89	0	4884	5042	5160
263/1559	210	2400.	3200.	0	89	0	4879	5051	5157
263/1706	211	2400.	3200.	0	89	0	4878	5059	5150
263/1758	212	2400.	3200.	0	89	0	4884	5056	5146
263/1900	213	2400.	3200.	0	89	0	4888	5044	5153
263/2003	214	2400.	3200.	0	89	0	4895	5030	5162
263/2111	215	2400.	3200.	0	89	0	4894	5031	5162
263/2200	216	2400.	3200.	0	89	0	4893	5026	5170
263/2310	217	2400.	3200.	0	89	0	4893	5024	5171
264/ 11	218	2400.	3200.	0	89	0	4893	5024	5173
264/ 106	219	2400.	3200.	0	89	0	4889	5028	5171
264/ 202	220	2400.	3200.	0	89	0	4884	5041	5165
264/ 255	221	2400.	3200.	0	89	0	4884	5045	5157
264/ 817	222	2400.	3200.	0	89	0	4887	5040	5159
264/ 912	223	2400.	3200.	0	89	0	4892	5031	5163
264/1005	224	2400.	3200.	0	89	0	4898	5024	5163
264/1056	225	2400.	3200.	0	89	0	4902	5020	5163
264/1202	226	2400.	3200.	0	89	0	4898	5022	5165
264/1308	227	2400.	3200.	0	89	0	4896	5028	5161
264/1404	228	2400.	3200.	0	89	0	4891	5034	5162
264/1455	229	2400.	3200.	0	89	0	4886	5042	5158
264/1601	230	2400.	3200.	0	89	0	4887	5039	5158
264/1701	231	2400.	3200.	0	89	0	4889	5050	5148
264/1804	232	2400.	3200.	0	89	0	4888	5055	5141
264/1900	233	2400.	3200.	0	89	0	4890	5043	5152
264/2000	234	2400.	3200.	0	89	0	4892	5035	5160
264/2107	235	2400.	3200.	0	89	0	4895	5029	5163
264/2203	236	2400.	3200.	0	89	0	4905	5020	5161
264/2233	237	2400.	3200.	0	89	0	4905	5018	5161
264/2316	238	2400.	3200.	0	89	0	4906	5018	5161
265/ 31	239	2400.	3200.	0	89	0	4902	5022	5163
265/ 125	240	2400.	3200.	0	89	0	4900	5023	5165
265/ 205	241	2400.	3200.	0	89	0	4891	5033	5163
265/ 300	242	2400.	3200.	0	89	0	4895	5032	5159
265/ 400	243	2400.	3200.	0	89	0	4894	5032	5163
265/ 500	244	2400.	3200.	0	89	0	4889	5040	5157
265/ 602	245	2400.	3200.	0	89	0	4888	5039	5162
265/ 704	246	2400.	3200.	0	89	0	4889	5039	5161
265/ 804	247	2400.	3200.	0	89	0	4890	5035	5163
265/ 906	248	2400.	3200.	0	89	0	4896	5026	5165
265/1000	249	2400.	3200.	0	89	0	4900	5019	5167
265/1106	250	2400.	3200.	0	89	0	4904	5013	5169
265/1306	251	2400.	3200.	0	89	0	4901	5014	5171
265/1355	252	2400.	3200.	0	89	0	4897	5023	5167
265/1502	253	2400.	3200.	0	89	0	4891	5034	5164
265/1534	254	2400.	3200.	0	89	0	4892	5033	5161
265/1609	255	2400.	3200.	0	89	0	4888	5038	5161
265/1659	256	2400.	3200.	0	89	0	4888	5042	5157
265/1800	257	2400.	3200.	0	89	0	4890	5040	5154

Dec 21 20:56 1988 AppA.22.doc Page 12

265/1900	258	2400.	3200.	0	89	0	4888	5047	5150
265/2004	259	2400.	3200.	0	89	0	4885	5042	5159
265/2117	260	2400.	3200.	0	89	0	4888	5039	5160
265/2210	261	2400.	3200.	0	89	0	4893	5033	5161
265/2302	262	2400.	3200.	0	89	0	4899	5023	5165
266/ 17	263	2400.	3200.	0	89	0	4894	5032	5159
266/ 103	264	2400.	3200.	0	89	0	4894	5030	5162
2[illegible]/ 159	265	2400.	3200.	0	89	0	4891	5034	5162
[illegible]6/ 252	266	2400.	3200.	0	89	0	4882	5046	5158
2[illegible]/ 4[illegible]1	267	2400.	3200.	0	89	0	4886	5047	5153
26[illegible]/ 5[illegible]	268	2400.	3200.	0	89	0	4881	5055	5150
266/ [illegible]	269	2400.	3200.	0	89	0	4884	5055	5146
266/ 7[illegible]	270	2400.	3200.	0	89	0	4886	5052	5148
266/ 859	271	2400.	3200.	0	89	0	4887	5047	5151
266/ 901	272	2400.	3200.	0	89	0	4894	5030	5161
266/1002	273	2400.	3200.	0	89	0	4893	5028	5166
266/1104	274	2400.	3200.	0	89	0	4897	5030	5157
266/1200	275	2400.	3200.	0	89	0	4900	5031	5153
266/1306	276	2400.	3200.	0	89	0	4901	5030	5153
266/1404	277	2400.	3200.	0	89	0	4901	5030	5152
266/1450	278	2400.	3200.	0	89	0	4896	5042	5145
266/1602	279	2400.	3200.	0	89	0	4893	5050	5140
266/1700	280	2400.	3200.	0	89	0	4889	5056	5138
266/1800	281	2400.	3200.	0	89	0	4892	5056	5136
266/1900	282	2400.	3200.	0	89	0	4893	5052	5139
266/1959	283	2400.	3200.	0	89	0	4897	5044	5143
266/2004	284	2400.	3200.	0	89	0	4897	5044	5143
266/2100	285	2400.	3200.	0	89	0	4898	5040	5145
266/2302	286	2400.	3200.	0	89	0	4904	5034	5145
267/ 10	287	2400.	3200.	0	89	0	4906	5030	5148
267/ 112	288	2400.	3200.	0	89	0	4901	5035	5147
267/ 159	289	2400.	3200.	0	89	0	4896	5046	5141
267/ 300	290	2400.	3200.	0	89	0	4893	5052	5138
267/ 400	291	2400.	3200.	0	89	0	4891	5056	5135
267/ 501	292	2400.	3200.	0	89	0	4889	5061	5133
267/ 559	293	2400.	3200.	0	89	0	4890	5063	5129
267/ 702	294	2400.	3200.	0	89	0	4894	5055	5134
267/ 806	295	2400.	3200.	0	89	0	4893	5056	5134
267/ 902	296	2400.	3200.	0	89	0	4895	5051	5138
267/1002	297	2400.	3200.	0	89	0	4900	5045	5138
267/1102	298	2400.	3200.	0	89	0	4897	5043	5142
267/1210	299	2400.	3200.	0	89	0	4899	5044	5140
267/1311	300	2400.	3200.	0	89	0	4900	5045	5137
267/1358	301	2400.	3200.	0	89	0	4897	5048	5137
267/1503	302	2400.	3200.	0	89	0	4891	5060	5131
267/1600	303	2400.	3200.	0	89	0	4885	5074	5123
267/1703	304	2400.	3200.	0	89	0	4877	5099	5107
267/1800	305	2400.	3200.	0	89	0	4878	5091	5115
267/1900	306	2400.	3200.	0	89	0	4885	5082	5115
267/2004	307	2400.	3200.	0	89	0	4902	5058	5122
267/2100	308	2400.	3200.	0	89	0	4911	5046	5127
267/2201	309	2400.	3200.	0	89	0	4904	5045	5133
268/ 102	310	2400.	3200.	0	89	0	4897	5050	5133
268/ 303	311	2400.	3200.	0	89	0	4899	5047	5134
268/ 425	312	2400.	3200.	0	89	0	4897	5046	5140
268/ 501	313	2400.	3200.	0	89	0	4899	5045	5139

268/ 600	314	2400.	3200.	0	89	0	4901	5049	5134
268/ 659	315	2400.	3200.	0	89	0	4907	5041	5134
268/ 800	316	2400.	3200.	0	89	0	4908	5037	5138
268/ 859	317	2400.	3200.	0	89	0	4910	5032	5142
268/1001	318	2400.	3200.	0	89	0	4913	5029	5141
268/1223	319	2400.	3200.	0	89	0	4908	5025	5152
268/1302	320	2400.	3200.	0	89	0	4902	5028	5154
268/1405	321	2400.	3200.	0	89	0	4894	5045	5144
268/1434	322	2400.	3200.	0	89	0	4895	5050	5138
268/1504	323	2400.	3200.	0	89	0	4893	5050	5138
268/1601	324	2400.	3200.	0	89	0	4891	5054	5137
268/1703	325	2400.	3200.	0	89	0	4886	5071	5125
268/1802	326	2400.	3200.	0	89	0	4890	5075	5118
268/1900	327	2400.	3200.	0	89	0	4907	5047	5128
268/2001	328	2400.	3200.	0	89	0	4917	5020	5150
268/2105	329	2400.	3200.	0	89	0	4920	5014	5152
268/2159	330	2400.	3200.	0	89	0	4921	5002	5166
268/2301	331	2400.	3200.	0	89	0	4918	4998	5176
269/ 1	332	2400.	3200.	0	89	0	4922	4993	5175
269/ 59	333	2400.	3200.	0	89	0	4922	4995	5171
269/ 200	334	2400.	3200.	0	89	0	4914	5007	5164
269/ 301	335	2400.	3200.	0	89	0	4910	5015	5160
269/ 400	336	2400.	3200.	0	89	0	4908	5021	5155
269/ 500	337	2400.	3200.	0	89	0	4902	5038	5142
269/ 601	338	2400.	3200.	0	89	0	4899	5048	5133
269/ 701	339	2400.	3200.	0	89	0	4905	5046	5131
269/ 804	340	2400.	3200.	0	89	0	4913	5030	5139
269/ 903	341	2400.	3200.	0	89	0	4915	5028	5141
269/1000	342	2400.	3200.	0	89	0	4918	5010	5160
269/1112	343	2400.	3200.	0	89	0	4912	5016	5156
269/1200	344	2400.	3200.	0	89	0	4914	5012	5161
269/1257	345	2400.	3200.	0	89	0	4923	4990	5177
269/1357	346	2400.	3200.	0	89	0	4922	4980	5193
269/1500	347	2400.	3200.	0	89	0	4915	4985	5194
269/1600	348	2400.	3200.	0	89	0	4912	4989	5193
269/1720	349	2400.	3200.	0	89	0	4904	5009	5177
269/1800	350	2400.	3200.	0	89	0	4908	5004	5181
269/1900	351	2400.	3200.	0	89	0	4908	5017	5164
269/1957	352	2400.	3200.	0	89	0	4916	5006	5169
269/2102	353	2400.	3200.	0	89	0	4920	4993	5181
269/2159	354	2400.	3200.	0	89	0	4921	4983	5192
269/2303	355	2400.	3200.	0	89	0	4925	4973	5202
270/ 44	356	2400.	3200.	0	89	0	4925	4975	5196
270/ 55	357	2400.	3200.	0	89	0	4925	4977	5193
270/ 131	358	2400.	3200.	0	89	0	4926	4973	5197
270/ 134	359	2400.	3200.	0	89	0	4929	4970	5198
270/ 208	360	2400.	3200.	0	89	0	4925	4969	5203
270/ 259	361	2400.	3200.	0	89	0	4916	4975	5207
270/ 402	362	2400.	3200.	0	89	0	4912	4990	5192
270/ 500	363	2400.	3200.	0	89	0	4911	5003	5176
270/ 600	364	2400.	3200.	0	89	0	4907	5026	5150
270/ 700	365	2400.	3200.	0	89	0	4920	5018	5147
270/ 801	366	2400.	3200.	0	89	0	4932	5000	5155
270/ 901	367	2400.	3200.	0	89	0	4943	4975	5175
270/1002	368	2400.	3200.	0	89	0	4936	4972	5185
270/1103	369	2400.	3200.	0	89	0	4942	4958	5198

270/1155	370	2400.	3200.	0	89	0	4939	4957	5203
270/1259	371	2400.	3200.	0	89	0	4932	4956	5215
270/1325	372	2400.	3200.	0	89	0	4925	4959	5222
270/1422	373	2400.	3200.	0	89	0	4915	4960	5230
270/1501	374	2400.	3200.	0	89	0	4907	4973	5223
270/1600	375	2400.	3200.	0	89	0	4911	4980	5209
270/1633	376	2400.	3200.	0	89	0	4903	4989	5207
270/1700	377	2400.	3200.	0	89	0	4906	4995	5194
270/1735	378	2400.	3200.	0	89	0	4904	5005	5178
270/1803	379	2400.	3200.	0	89	0	4909	5008	5173
270/1900	380	2400.	3200.	0	89	0	4910	5011	5167
270/2002	381	2400.	3200.	0	89	0	4906	5021	5161
270/2101	382	2400.	3200.	0	89	0	4911	5013	5164
270/2201	383	2400.	3200.	0	89	0	4918	5002	5170
271/ 230	384	2400.	3200.	0	89	0	4914	4991	5191
271/ 304	385	2400.	3200.	0	89	0	4914	4994	5183
271/ 400	386	2400.	3200.	0	89	0	4913	5001	5173
271/ 500	387	2400.	3200.	0	89	0	4907	5002	5180
271/ 600	388	2400.	3200.	0	89	0	4909	5010	5168
271/ 707	389	2400.	3200.	0	89	0	4906	5021	5159
271/ 803	390	2400.	3200.	0	89	0	4905	5026	5152
271/ 900	391	2400.	3200.	0	89	0	4911	5017	5158
271/1002	392	2400.	3200.	0	89	0	4914	5009	5163
271/1109	393	2400.	3200.	0	89	0	4912	5011	5165
271/1209	394	2400.	3200.	0	89	0	4910	5007	5171
271/1358	395	2400.	3200.	0	89	0	4902	5016	5172
271/1501	396	2400.	3200.	0	89	0	4895	5017	5178
271/1603	397	2400.	3200.	0	89	0	4896	5020	5174
271/1858	398	2400.	3200.	0	89	0	4886	5053	5147
271/1917	399	2400.	3200.	0	89	0	4886	5055	5146
271/1931	400	2400.	3200.	0	89	0	4887	5052	5146
271/1945	401	2400.	3200.	0	89	0	4887	5056	5143
271/2001	402	2400.	3200.	0	89	0	4885	5061	5140
271/2016	403	2400.	3200.	0	89	0	4886	5057	5141
271/2030	404	2400.	3200.	0	89	0	4887	5060	5139
271/2046	405	2400.	3200.	0	89	0	4881	5062	5139
271/2102	406	2400.	3200.	0	89	0	4886	5059	5140
271/2118	407	2400.	3200.	0	89	0	4890	5056	5138
271/2134	408	2400.	3200.	0	89	0	4896	5047	5142
271/2148	409	2400.	3200.	0	89	0	4900	5043	5142
271/2201	410	2400.	3200.	0	89	0	4902	5042	5140
271/2217	411	2400.	3200.	0	89	0	4902	5042	5139
271/2230	412	2400.	3200.	0	89	0	4903	5039	5143
271/2245	413	2400.	3200.	0	89	0	4900	5037	5148
271/2259	414	2400.	3200.	0	89	0	4902	5032	5152
271/2316	415	2400.	3200.	0	89	0	4903	5031	5151
272/ 127	416	2400.	3200.	0	89	0	4909	5013	5164
272/ 201	417	2400.	3200.	0	89	0	4909	5012	5166
272/ 300	418	2400.	3200.	0	89	0	4904	5014	5169
272/ 400	419	2400.	3200.	0	89	0	4901	5007	5182
272/ 500	420	2400.	3200.	0	89	0	4896	5018	5173
272/ 600	421	2400.	3200.	0	89	0	4899	5024	5163
272/ 658	422	2400.	3200.	0	89	0	4897	5035	5151
272/ 905	423	2400.	3200.	0	89	0	4902	5040	5140
272/1043	424	2400.	3200.	0	89	0	4898	5036	5150

TRS.trs:

Display model

Red	591.	4158.	4564.	transponder 1
Grn	2600.	797.	4566.	transponder 2
Blu	4428.	4771.	4573.	transponder 3

---

xxx.pos:

2	2470.	3282.	0.0	89.	0.	4906.	5086.	5087.
4	2479.	3256.	0.4	89.	0.	4914.	5074.	5091.
6	2502.	3220.	0.3	89.	0.	4929.	5056.	5093.
13	2465.	3167.	0.1	89.	0.	4925.	5033.	5123.
14	2437.	3164.	0.8	89.	0.	4915.	5032.	5134.
17	2454.	3258.	0.2	89.	0.	4904.	5075.	5100.
18	2459.	3240.	0.8	89.	0.	4909.	5066.	5103.
19	2453.	3239.	0.3	89.	0.	4907.	5066.	5106.
20	2462.	3177.	0.4	89.	0.	4922.	5037.	5121.
21	2455.	3185.	0.3	89.	0.	4918.	5041.	5121.
22	2467.	3180.	0.9	89.	0.	4923.	5038.	5118.
23	2441.	3185.	0.7	89.	0.	4913.	5042.	5127.
25	2443.	3202.	1.0	89.	0.	4910.	5049.	5120.
29	2455.	3214.	1.2	89.	0.	4912.	5054.	5112.
30	2449.	3207.	0.7	89.	0.	4912.	5052.	5117.
31	2438.	3225.	0.8	89.	0.	4904.	5060.	5115.
33	2442.	3223.	0.0	89.	0.	4906.	5059.	5115.
34	2441.	3232.	0.7	89.	0.	4904.	5063.	5112.
35	2431.	3238.	1.2	89.	0.	4900.	5067.	5115.
36	2426.	3244.	1.1	89.	0.	4897.	5070.	5115.
37	2415.	3276.	0.5	89.	0.	4887.	5085.	5110.
38	2422.	3263.	0.9	89.	0.	4892.	5079.	5111.
39	2419.	3250.	0.8	89.	0.	4893.	5073.	5116.
40	2423.	3231.	0.9	89.	0.	4898.	5064.	5120.
41	2424.	3223.	0.9	89.	0.	4900.	5060.	5122.
42	2426.	3231.	1.0	89.	0.	4899.	5064.	5119.
44	2426.	3244.	1.1	89.	0.	4897.	5070.	5115.
45	2425.	3253.	1.1	89.	0.	4895.	5074.	5113.
46	2422.	3262.	0.2	89.	0.	4892.	5078.	5111.
47	2431.	3251.	0.5	89.	0.	4897.	5073.	5111.
48	2441.	3216.	0.1	89.	0.	4907.	5056.	5117.
49	2418.	3227.	0.7	89.	0.	4897.	5062.	5123.
50	2413.	3235.	1.0	89.	0.	4893.	5065.	5122.
52	2421.	3280.	0.7	89.	0.	4888.	5086.	5106.
54	2424.	3258.	0.5	89.	0.	4893.	5076.	5111.
55	2433.	3259.	0.9	89.	0.	4896.	5076.	5107.
57	2431.	3249.	0.9	89.	0.	4897.	5071.	5111.
58	2440.	3243.	1.4	89.	0.	4901.	5068.	5109.
59	2427.	3258.	0.4	89.	0.	4894.	5076.	5110.
61	2423.	3250.	0.5	89.	0.	4894.	5072.	5114.
62	2408.	3259.	0.4	89.	0.	4887.	5077.	5117.
63	2408.	3295.	0.0	89.	0.	4881.	5094.	5107.
64	2410.	3281.	0.5	89.	0.	4884.	5087.	5110.

65	2414.	3272.	0.2	89.	0.	4887.	5083.	5111.
66	2422.	3275.	0.7	89.	0.	4889.	5084.	5107.
67	2426.	3292.	0.0	89.	0.	4888.	5092.	5101.
68	2441.	3270.	0.7	89.	0.	4897.	5081.	5101.
69	2453.	3209.	0.9	89.	0.	4913.	5053.	5115.
70	2436.	3203.	0.3	89.	0.	4908.	5050.	5123.
71	2431.	3212.	0.5	89.	0.	4904.	5054.	5122.
72	2414.	3218.	0.4	89.	0.	4897.	5058.	5127.
77	2423.	3233.	0.6	89.	0.	4897.	5064.	5119.
78	2419.	3203.	1.1	89.	0.	4901.	5050.	5129.
79	2418.	3218.	0.9	89.	0.	4898.	5057.	5125.
80	2412.	3202.	0.7	89.	0.	4899.	5050.	5132.
81	2427.	3174.	0.1	89.	0.	4910.	5037.	5135.
82	2410.	3136.	0.7	89.	0.	4911.	5020.	5153.
84	2347.	3117.	0.7	89.	0.	4893.	5015.	5184.
85	2335.	3125.	0.2	89.	0.	4887.	5019.	5186.
86	2341.	3141.	0.6	89.	0.	4886.	5026.	5179.
87	2334.	3164.	0.4	89.	0.	4879.	5036.	5174.
88	2357.	3149.	0.6	89.	0.	4890.	5029.	5170.
89	2360.	3131.	1.5	89.	0.	4895.	5021.	5175.
91	2343.	3102.	0.3	89.	0.	4895.	5008.	5190.
93	2365.	3122.	0.1	89.	0.	4898.	5016.	5175.
94	2372.	3131.	0.7	89.	0.	4899.	5020.	5170.
95	2364.	3085.	1.3	89.	0.	4906.	5000.	5188.
96	2339.	3039.	0.7	89.	0.	4907.	4981.	5212.
102	2371.	3060.	0.6	89.	0.	4913.	4988.	5192.
103	2366.	3068.	0.3	89.	0.	4910.	4992.	5192.
104	2364.	3065.	0.7	89.	0.	4910.	4991.	5194.
105	2367.	3077.	1.2	89.	0.	4908.	4995.	5188.
106	2365.	3085.	0.7	89.	0.	4906.	5000.	5187.
107	2355.	3095.	0.8	89.	0.	4900.	5004.	5187.
108	2359.	3128.	0.5	89.	0.	4895.	5019.	5176.
109	2364.	3128.	1.4	89.	0.	4896.	5018.	5173.
110	2370.	3103.	0.1	89.	0.	4904.	5007.	5179.
111	2368.	3103.	0.3	89.	0.	4903.	5007.	5180.
112	2384.	3110.	1.1	89.	0.	4907.	5009.	5171.
113	2397.	3111.	0.4	89.	0.	4912.	5010.	5166.
114	2387.	3108.	0.4	89.	0.	4909.	5009.	5171.
115	2391.	3072.	0.6	89.	0.	4918.	4993.	5181.
116	2392.	3084.	0.7	89.	0.	4916.	4998.	5177.
118	2381.	3066.	1.1	89.	0.	4916.	4991.	5187.
119	2380.	3066.	0.5	89.	0.	4915.	4990.	5187.
120	2342.	3064.	1.4	89.	0.	4903.	4992.	5203.
121	2320.	3074.	0.5	89.	0.	4893.	4997.	5208.
122	2336.	3102.	0.9	89.	0.	4892.	5008.	5192.
123	2353.	3109.	1.6	89.	0.	4896.	5010.	5183.
125	2364.	3115.	1.3	89.	0.	4899.	5012.	5177.
126	2365.	3102.	0.4	89.	0.	4902.	5007.	5181.
127	2366.	3128.	1.2	89.	0.	4897.	5018.	5172.
128	2377.	3157.	0.8	89.	0.	4895.	5031.	5159.
129	2392.	3159.	0.7	89.	0.	4900.	5031.	5153.
130	2394.	3143.	0.4	89.	0.	4904.	5024.	5157.
131	2395.	3137.	0.9	89.	0.	4906.	5022.	5159.
132	2392.	3142.	0.9	89.	0.	4904.	5024.	5159.
133	2375.	3109.	1.0	89.	0.	4904.	5009.	5175.
134	2373.	3129.	0.7	89.	0.	4900.	5019.	5170.

Dec 21 20:56 1988 AppA.22.doc Page 17

135	2368.	3134.	0.1	89.	0.	4897.	5021.	5170.
136	2376.	3147.	0.2	89.	0.	4897.	5027.	5163.
137	2376.	3147.	0.2	89.	0.	4897.	5027.	5163.
138	2382.	3153.	0.1	89.	0.	4898.	5029.	5159.
139	2370.	3136.	0.1	89.	0.	4897.	5022.	5169.
140	2373.	3141.	0.2	89.	0.	4897.	5024.	5166.
141	2372.	3145.	0.1	89.	0.	4896.	5026.	5165.
142	2378.	3162.	0.9	89.	0.	4895.	5034.	5158.
143	2377.	3148.	0.4	89.	0.	4897.	5027.	5162.
144	2371.	3160.	0.5	89.	0.	4893.	5033.	5161.
145	2374.	3160.	0.6	89.	0.	4894.	5033.	5160
148	2379.	3155.	0.8	89.	0.	4896.	5030.	5159.
150	2385.	3166.	1.5	89.	0.	4896.	5034.	5153.
154	2386.	3152.	1.2	89.	0.	4899.	5028.	5157.
157	2378.	3158.	1.4	89.	0.	4895.	5031.	5158.
158	2373.	3171.	0.2	89.	0.	4891.	5038.	5157.
159	2373.	3188.	0.7	89.	0.	4888.	5046.	5152.
160	2378.	3201.	0.3	89.	0.	4887.	5051.	5146.
161	2375.	3193.	0.9	89.	0.	4888.	5048.	5150.
162	2370.	3197.	0.0	89.	0.	4885.	5050.	5150.
163	2368.	3196.	1.3	89.	0.	4885.	5050.	5152.
164	2372.	3185.	0.2	89.	0.	4888.	5044.	5153.
165	2374.	3189.	0.0	89.	0.	4888.	5046.	5151.
166	2376.	3192.	0.5	89.	0.	4888.	5047.	5149.
167	2381.	3181.	0.2	89.	0.	4892.	5042.	5151.
168	2382.	3170.	0.7	89.	0.	4895.	5037.	5154.
169	2369.	3165.	0.2	89.	0.	4891.	5035.	5160.
170	2363.	3184.	0.1	89.	0.	4885.	5044.	5157.
171	2369.	3180.	0.2	89.	0.	4888.	5042.	5156.
172	2373.	3188.	0.7	89.	0.	4888.	5046.	5152.
173	2370.	3200.	1.5	89.	0.	4885.	5052.	5150.
175	2368.	3190.	0.4	89.	0.	4886.	5047.	5153.
176	2374.	3179.	0.3	89.	0.	4890.	5041.	5154.
177	2372.	3174.	0.5	89.	0.	4890.	5039.	5156.
178	2375.	3169.	1.2	89.	0.	4892.	5036.	5156.
182	2367.	3179.	1.4	89.	0.	4887.	5041.	5156.
183	2370.	3186.	1.0	89.	0.	4887.	5044.	5153.
184	2379.	3184.	1.3	89.	0.	4890.	5043.	5150.
185	2371.	3188.	0.9	89.	0.	4887.	5045.	5152.
186	2374.	3196.	0.2	89.	0.	4887.	5049.	5149.
187	2370.	3199.	0.7	89.	0.	4885.	5051.	5150.
188	2366.	3204.	1.4	89.	0.	4883.	5054.	5150.
189	2367.	3208.	0.0	89.	0.	4882.	5055.	5148.
190	2369.	3192.	0.6	89.	0.	4886.	5048.	5152.
191	2371.	3189.	0.2	89.	0.	4887.	5046.	5152.
192	2375.	3171.	0.3	89.	0.	4892.	5038.	5156.
193	2373.	3159.	1.0	89.	0.	4893.	5032.	5160.
194	2372.	3149.	0.1	89.	0.	4895.	5028.	5164.
195	2362.	3139.	0.5	89.	0.	4894.	5024.	5171.
196	2361.	3167.	1.4	89.	0.	4887.	5036.	5162.
197	2356.	3164.	1.1	89.	0.	4887.	5036.	5166.
198	2350.	3178.	0.4	89.	0.	4882.	5042.	5164.
200	2360.	3174.	0.2	89.	0.	4886.	5040.	5161.
201	2352.	3162.	0.8	89.	0.	4886.	5035.	5168.
202	2356.	3162.	0.4	89.	0.	4887.	5034.	5166.
203	2369.	3168.	0.9	89.	0.	4890.	5036.	5159.

Dec 21 20:56 1988 AppA.22.doc Page 18

206	2367.	3163.	1.2	89.	0.	4890.	5034.	5161.
207	2366.	3169.	0.3	89.	0.	4889.	5037.	5160:
208	2364.	3172.	0.4	89.	0.	4888.	5039.	5160.
209	2358.	3179.	0.5	89.	0.	4884.	5042.	5160.
210	2353.	3198.	0.3	89.	0.	4879.	5051.	5157.
211	2358.	3215.	1.0	89.	0.	4878.	5059.	5150.
212	2372.	3210.	1.1	89.	0.	4884.	5056.	5146.
213	2372.	3185.	0.2	89.	0.	4888.	5044.	5153.
214	2374.	3154.	0.4	89.	0.	4895.	5030.	5162.
215	2372.	3156.	0.3	89.	0.	4894.	5031.	5162.
216	2362.	3143.	0.7	89.	0.	4893.	5026.	5170.
217	2361.	3140.	0.3	89.	0.	4893.	5024.	5171.
218	2359.	3138.	1.0	89.	0.	4893.	5024.	5173.
219	2354.	3148.	0.3	89.	0.	4889.	5028.	5171.
221	2361.	3186.	0.1	89.	0.	4884.	5045.	5157.
222	2364.	3175.	0.2	89.	0.	4887.	5040.	5159.
223	2368.	3156.	0.6	89.	0.	4892.	5031.	5163.
224	2378.	3143.	1.3	89.	0.	4898.	5024.	5163.
225	2384.	3134.	1.3	89.	0.	4902.	5020.	5163.
227	2377.	3151.	1.0	89.	0.	4896.	5028.	5161.
228	2367.	3162.	0.3	89.	0.	4891.	5034.	5162.
229	2363.	3180.	0.2	89.	0.	4886.	5042.	5158.
232	2384.	3209.	0.3	89.	0.	4888.	5055.	5141.
233	2376.	3183.	0.0	89.	0.	4890.	5043.	5152.
234	2371.	3164.	0.6	89.	0.	4892.	5035.	5160.
235	2373.	3152.	0.2	89.	0.	4895.	5029.	5163.
236	2391.	3134.	0.2	89.	0.	4905.	5020.	5161.
238	2393.	3130.	1.0	89.	0.	4906.	5018.	5161.
239	2384.	3137.	0.3	89.	0.	4902.	5022.	5163.
240	2379.	3138.	0.8	89.	0.	4900.	5023.	5165.
241	2366.	3160.	0.1	89.	0.	4891.	5033.	5163.
242	2377.	3159.	0.0	89.	0.	4895.	5032.	5159.
244	2369.	3175.	0.1	89.	0.	4889.	5040.	5157.
247	2364.	3163.	0.9	89.	0.	4890.	5035.	5163.
248	2372.	3145.	0.1	89.	0.	4896.	5026.	5165.
249	2377.	3131.	1.1	89.	0.	4900.	5019.	5167.
250	2381.	3119.	1.5	89.	0.	4904.	5013.	5169.
252	2372.	3139.	0.4	89.	0.	4897.	5023.	5167.
254	2370.	3161.	0.3	89.	0.	4892.	5033.	5161.
255	2363.	3170.	0.3	89.	0.	4888.	5038.	5161.
256	2367.	3179.	0.8	89.	0.	4888.	5042.	5157.
257	2374.	3177.	1.0	89.	0.	4890.	5040.	5154.
258	2375.	3191.	0.1	89.	0.	4888.	5047.	5150.
259	2361.	3179.	0.3	89.	0.	4885.	5042.	5159.
260	2364.	3172.	0.4	89.	0.	4888.	5039.	5160.
261	2371.	3160.	0.5	89.	0.	4893.	5033.	5161.
262	2377.	3139.	0.0	89.	0.	4899.	5023.	5165.
263	2375.	3159.	0.8	89.	0.	4894.	5032.	5159.
264	2372.	3154.	0.4	89.	0.	4894.	5030.	5162.
265	2367.	3162.	0.3	89.	0.	4891.	5034.	5162.
266	2357.	3188.	0.4	89.	0.	4882.	5046.	5158.
267	2368.	3190.	0.4	89.	0.	4886.	5047.	5153.
268	2363.	3207.	0.5	89.	0.	4881.	5055.	5150.
269	2373.	3208.	0.3	89.	0.	4884.	5055.	5146.
270	2374.	3201.	1.0	89.	0.	4886.	5052.	5148.
271	2372.	3191.	0.0	89.	0.	4887.	5047.	5151.

Dec 21 20:56 1988 AppA.22.doc Page 19

272	2373.	3155.	1.1	89.	0.	4894.	5030.	5161.
273	2366.	3149.	0.3	89.	0.	4893.	5028.	5166.
274	2383.	3156.	1.2	89.	0.	4897.	5030.	5157.
275	2392.	3159.	0.7	89.	0.	4900.	5031.	5153.
276	2393.	3157.	0.7	89.	0.	4901.	5030.	5153.
277	2395.	3157.	1.3	89.	0.	4901.	5030.	5152.
278	2394.	3183.	0.5	89.	0.	4896.	5042.	5145.
279	2394.	3200.	0.1	89.	0.	4893.	5050.	5140.
280	2389.	3212.	0.1	89.	0.	4889.	5056.	5138.
281	2396.	3212.	0.9	89.	0.	4892.	5056.	5136.
282	2395.	3203.	0.7	89.	0.	4893.	5052.	5139.
283	2397.	3186.	0.5	89.	0.	4897.	5044.	5143.
284	2397.	3186.	0.5	89.	0.	4897.	5044.	5143.
285	2397.	3179.	0.5	89.	0.	4898.	5040.	5145.
286	2407.	3166.	0.4	89.	0.	4904.	5034.	5145.
287	2407.	3157.	0.0	89.	0.	4906.	5030.	5148.
288	2400.	3168.	0.7	89.	0.	4901.	5035	5147.
289	2398.	3191.	0.1	89.	0.	4896.	5046.	5141.
290	2396.	3204.	0.1	89.	0.	4893.	5052.	5138.
291	2396.	3213.	0.5	89.	0.	4891.	5056.	5135.
292	2395.	3223.	0.3	89.	0.	4889.	5061.	5133.
293	2400.	3228.	0.1	89.	0.	4890.	5063.	5129.
294	2402.	3211.	0.5	89.	0.	4894.	5055.	5134.
295	2400.	3213.	0.4	89.	0.	4893.	5056.	5134.
296	2399.	3201.	0.9	89.	0.	4895.	5051.	5138.
297	2408.	3190.	0.3	89.	0.	4900.	5045.	5138.
298	2399.	3186.	1.0	89.	0.	4897.	5043.	5142.
299	2404.	3187.	0.1	89.	0.	4899.	5044.	5140.
300	2409.	3190.	0.4	89.	0.	4900.	5045.	5137.
301	2404.	3197.	0.5	89.	0.	4897.	5048.	5137.
302	2400.	3222.	0.2	89.	0.	4891.	5060.	5131.
303	2398.	3252.	0.1	89.	0.	4885.	5074.	5123.
304	2401.	3304.	0.2	89.	0.	4877.	5099.	5107.
305	2395.	3286.	0.9	89.	0.	4878.	5091.	5115.
306	2407.	3269.	0.2	89.	0.	4885.	5082.	5115.
307	2427.	3219.	0.9	89.	0.	4902.	5058.	5122.
309	2419.	3191.	0.1	89.	0.	4904.	5045.	5133.
312	2400.	3191.	0.0	89.	0.	4897.	5046.	5140.
313	2405.	3189.	0.2	89.	0.	4899.	5045.	5139.
314	2413.	3197.	1.5	89.	0.	4901.	5049.	5134.
315	2423.	3182.	0.0	89.	0.	4907.	5041.	5134.
316	2421.	3173.	0.4	89.	0.	4908.	5037.	5138.
317	2420.	3162.	0.7	89.	0.	4910.	5032.	5142.
318	2426.	3156.	0.0	89.	0.	4913.	5029.	5141.
319	2406.	3146.	0.3	89.	0.	4908.	5025.	5152.
320	2394.	3152.	0.8	89.	0.	4902.	5028.	5154.
321	2391.	3189.	0.5	89.	0.	4894.	5045.	5144.
322	2399.	3200.	0.1	89.	0.	4895.	5050.	5138.
323	2396.	3201.	1.5	89.	0.	4893.	5050.	5138.
324	2394.	3209.	0.7	89.	0.	4891.	5054.	5137.
325	2398.	3245.	0.2	89.	0.	4886.	5071.	5125.
326	2412.	3254.	1.2	89.	0.	4890.	5075.	5118.
327	2430.	3195.	0.6	89.	0.	4907.	5047.	5128.
329	2426.	3122.	0.7	89.	0.	4920.	5014.	5152.
330	2412.	3094.	0.8	89.	0.	4921.	5002.	5166.
332	2404.	3074.	0.1	89.	0.	4922.	4993.	5175.

Dec 21 20:56 1988 AppA.22.doc Page 20

333	2409.	3080.	1.0	89.	0.	4922.	4995.	5171.
335	2401.	3124.	1.0	89.	0.	4910.	5015.	5160.
336	2403.	3138.	1.0	89.	0.	4908.	5021.	5155.
337	2407.	3175.	0.9	89.	0.	4902.	5038.	5142.
338	2411.	3198.	1.5	89.	0.	4899.	5048.	5133.
339	2423.	3193.	0.3	89.	0.	4905.	5046.	5131.
340	2428.	3159.	0.5	89.	0.	4913.	5030.	5139.
341	2429.	3153.	0.8	89.	0.	4915.	5028.	5141.
342	2414.	3112.	1.2	89.	0.	4918.	5010.	5160.
343	2408.	3127.	1.2	89.	0.	4912.	5016.	5156.
344	2406.	3117.	0.4	89.	0.	4914.	5012.	5161.
345	2404.	3068.	0.6	89.	0.	4923.	4990.	5177.
346	2384.	3042.	0.2	89.	0.	4922.	4980.	5193.
347	2372.	3053.	0.4	89.	0.	4915.	4985.	5194.
348	2368.	3062.	0.0	89.	0.	4912.	4989.	5193.
349	2372.	3107.	0.3	89.	0.	4904.	5009.	5177.
354	2384.	3048.	1.4	89.	0.	4921.	4983.	5192.
356	2386.	3031.	0.1	89.	0.	4925.	4975.	5196.
357	2389.	3036.	0.0	89.	0.	4925.	4977.	5193.
358	2386.	3027.	0.2	89.	0.	4926.	4973.	5197.
359	2390.	3020.	0.1	89.	0.	4929.	4970.	5198.
360	2378.	3018.	0.8	89.	0.	4925.	4969.	5203.
361	2359.	3029.	0.2	89.	0.	4916.	4975.	5207.
362	2369.	3064.	0.2	89.	0.	4912.	4990.	5192.
363	2385.	3095.	0.3	89.	0.	4911.	5003.	5176.
364	2407.	3149.	1.0	89.	0.	4907.	5026.	5150.
365	2431.	3132.	0.7	89.	0.	4920.	5018.	5147.
366	2442.	3092.	0.4	89.	0.	4932.	5000.	5155.
367	2438.	3035.	0.5	89.	0.	4943.	4975.	5175.
368	2416.	3028.	1.0	89.	0.	4936.	4972.	5185.
369	2411.	2995.	0.6	89.	0.	4942.	4958.	5198.
370	2401.	2992.	0.6	89.	0.	4939.	4957.	5203.
371	2376.	2986.	0.4	89.	0.	4932.	4956.	5215.
373	2332.	2991.	0.5	89.	0.	4915.	4960.	5230.
374	2326.	3020.	0.4	89.	0.	4907.	4973.	5223.
375	2349.	3038.	1.3	89.	0.	4911.	4980.	5209.
376	2338.	3057.	1.3	89.	0.	4903.	4989.	5207.
377	2357.	3073.	0.9	89.	0.	4906.	4995.	5194.
379	2385.	3105.	0.9	89.	0.	4909.	5008.	5173.
380	2393.	3113.	0.3	89.	0.	4910.	5011.	5167.
381	2393.	3134.	1.3	89.	0.	4906.	5021.	5161.
382	2398.	3118.	0.8	89.	0.	4911.	5013.	5164.
383	2403.	3093.	1.1	89.	0.	4918.	5002.	5170.
385	2382.	3075.	0.4	89.	0.	4914.	4994.	5183.
387	2374.	3093.	1.2	89.	0.	4907.	5002.	5180.
388	2390.	3112.	0.6	89.	0.	4909.	5010.	5168.
389	2395.	3136.	0.1	89.	0.	4906.	5021.	5159.
390	2401.	3149.	1.3	89.	0.	4905.	5026.	5152.
391	2404.	3128.	0.1	89.	0.	4911.	5017.	5158.
392	2404.	3111.	0.8	89.	0.	4914.	5009.	5163.
393	2398.	3114.	0.6	89.	0.	4912.	5011.	5165.
394	2389.	3105.	0.4	89.	0.	4910.	5007.	5171.
395	2374.	3122.	1.2	89.	0.	4902.	5016.	5172.
396	2357.	3123.	0.2	89.	0.	4895.	5017.	5178.
397	2362.	3130.	0.9	89.	0.	4896.	5020.	5174.
398	2375.	3203.	1.1	89.	0.	4886.	5053.	5147.

400	2378.	3202.	0.5	89.	0.	4887.	5052.	5146.
403	2381.	3213.	0.2	89.	0.	4886.	5057.	5141.
405	2375.	3225.	1.5	89.	0.	4881.	5062.	5139.
406	2382.	3217.	1.0	89.	0.	4886.	5059.	5140.
407	2391.	3212.	0.7	89.	0.	4890.	5056.	5138.
408	2397.	3192.	1.3	89.	0.	4896.	5047.	5142.
409	2403.	3184.	1.4	89.	0.	4900.	5043.	5142.
410	2409.	3182.	0.9	89.	0.	4902.	5042.	5140.
411	2410.	3183.	0.2	89.	0.	4902.	5042.	5139.
412	2407.	3175.	1.4	89.	0.	4903.	5039.	5143.
413	2397.	3171.	0.7	89.	0.	4900.	5037.	5148.
414	2396.	3159.	1.0	89.	0.	4902.	5032.	5152.
415	2399.	3158.	0.4	89.	0.	4903.	5031.	5151.
416	2395.	3119.	0.8	89.	0.	4909.	5013.	5164.
417	2393.	3116.	0.3	89.	0.	4909.	5012.	5166.
418	2381.	3120.	0.7	89.	0.	4904.	5014.	5169.
419	2362.	3102.	0.6	89.	0.	4901.	5007.	5182.
420	2364.	3128.	1.4	89.	0.	4896.	5018.	5173.
421	2379.	3142.	0.5	89.	0.	4899.	5024.	5163.
422	2389.	3168.	1.2	89.	0.	4897.	5035.	5151.
423	2409.	3180.	0.6	89.	0.	4902.	5040.	5140.
424	2392.	3169.	0.3	89.	0.	4898.	5036.	5150.

APPENDIX B. Array Travel Time Acquisition Program Description.

HARRYNAV

Main Program	Subroutines Level 1	Level 2	Level 3
HARRYNAV			
	getnav	whichtp	printnavp
			printnavt
		skipread	whichtp
		gettime	
		skipread	
		dtfpos	
		skipread	
		fillnav	navloc
		dtfpos	
		skipread	

HARRYNAV: Enters user parameters from the program line. Parameters are the option flags, number of rollovers to skip between processing, total number of rollovers to process, rollover to start processing with, processor ID file, *FLIP* slant range data file name, and output file name. System subroutines are called to read the input program line and the processor ID file.

getnav: System subroutines are called to read the *FLIP* slant range data, parameters are initialized. Subroutines *whichtp* and *skipread* are called to read in the buffer of data containing the rollover requested. If the time flag is set, the first buffer on the tape is read, the header time parameter is converted to an integer and compared to the user input time parameter also converted to an integer (by *gettime*). Due to a lack of space in the tape header, only the least significant digit of the GMT hour was recorded so the local hour is converted to GMT for the correct GMT time. The time in each buffer header is compared until the header time in the rollover buffer is greater than the time requested. If the time flag is not set, buffers are read in until the rollover number requested is equal to the number of rollovers read in. A loop indexed by i over the number of rollovers requested is initialized. The file *window.dat* containing the valid window parameters is read in. *Dtfpos* is called to write the date and time into the output file and *FLIP* slant range into the output file. A loop for extracting the data for each transponder is initialized indexed by j. *Skipread* is called for the remaining three transponders. *Navtst1* is called to locate the return. Files are output containing the valid data markers (*pltwin.sio*) and the received data (*navplt.sio*) for plotting. Current valid data markers are written into *window.dat* and the date and time are appended to the bottom of the file by *dtfpos*. A system call to pltsav.scr plots the data to the screen. If pltsav.scr does not exist when called, a message is displayed and the program continues.

whichtp: Requests the user to load the magnetic tape and enter the tape

drive number. If the verbose flag is set, *printnavp* and *printnavt* are called.

printnavp: prints the array slant ranges by processor. printnavt: prints the array slant ranges by transponder.

skipread: Reads the first buffer requested. Buffers may be requested by rollover, by GMT time or by hardware clock time. If end of tape indicators are true then *whichtp* is called and the program continues extracting the navigation time series.

gettime: Translates the ascii string indicating GMT time into an integer.

dtfpos: is called to write the date and time into the output file and linearly interpolate the *FLIP* slant range data to obtain the slant range corresponding to the time of the rollover. The local date is converted to Julian day, and the time is GMT. If the passed parameter 'pall' is negative (0) then only the data and time are printed.

fillnav: For each transponder interrogation, the navigation bits are extracted from the data stream and stored as a time series for each processor. Each time series begins with the buffer following the interrogation transmission and ends 74 buffers later (9.472 s). Various ID's and counters are verified. Errors in the buffer counter, frame sync word or frame counter are fatal. If an error in *processor* ID is detected, the data for the processor in question is zero filled.

navloc: The data are received by navloc as the most significant 10 bits in a 16-bit word and repacked into the 30 least significant bits in a 32-bit word. The correlation is done as a moving adder with leading pointers i1 (word) and j1 (bit) and trailing pointers k (word) and l (bit) starting from the beginning of the plot window (bwinw[xpndr] + bwinb[xpndr] the window word and bit pointers) for SAVSZ bits (1 second of data). The plot window is set up as WINOFF words before the valid data window when the program is initiated and allowed to move as the reply moves by evaluating the average movement of the replies across the array. The detection is valid only within the valid data window, with parameters pstart, pend, LENGTH; and is defined as a normalized correlation amplitude of detec=1 (normalized by PINGSIZE). The time of the detection is the start of the plot window plus the number of bits, p, as the adder was initilized with the first PINGSIZE outside the correlation loop.

```
Dec 21 21:03 1988  AppB.11.doc Page 1


-------------------------------------------------------------

SAMPLE RUN of harrynav:


Script started on Wed Dec 21 10:38:59 1988
N17>harrynav
Usage: harrynav [-abptv] nskproll nroll t/lstroll procfile flipfile outnavfile
N18>harrynav 1 1 1 FLIPproc.new f out.858

Load tape(online!) Then enter 0,1 or q: 0

cntbuf: 180, start: 14
hwclock: 65522,  SEQN: 181
# Bad processor ID's = 1  Buffer# 183
# Bad processor ID's = 1  Buffer# 186
# Bad processor ID's = 1  Buffer# 194
# Bad processor ID's = 1  Buffer# 228
# Bad processor ID's = 1  Buffer# 235
# Bad processor ID's = 1  Buffer# 237
iadd=0

Roll: 1,  Xponder: 1  Start: 14
Proc# 0;  5822.120117  8733.179688  8
Proc# 1;  5865.719727  8798.580078  8
Proc# 2;  5909.319824  8863.979492  8
Proc# 3;  5953.319824  8929.979492  8
Proc# 4;  5998.120117  8997.179688  5
Proc# 5;  6042.520020  9063.780273  7
Proc# 6;  6087.319824  9130.979492  4
Proc# 7;  6132.520020  9198.780273  4
Proc# 8;  6177.719727  9266.580078  6
Proc# 9;  6222.919922  9334.379883  7
Proc# 10;  6269.719727  9404.580078  5
Proc# 11;  6314.120117  9471.179688  7

cntbuf: 258, start: 30
hwclock: 9970,  SEQN: 259
# Bad processor ID's = 1  Buffer# 259
# Bad processor ID's = 1  Buffer# 270
# Bad processor ID's = 1  Buffer# 276
# Bad processor ID's = 2  Buffer# 303
# Bad processor ID's = 1  Buffer# 317
# Bad processor ID's = 1  Buffer# 325
# Bad processor ID's = 1  Buffer# 328
iadd=0

Roll: 1,  Xponder: 2  Start: 30
Proc# 0;  5983.319824  8974.979492  9
Proc# 1;  6025.319824  9037.979492  12
Proc# 2;  6067.719727  9101.580078  9
Proc# 3;  6110.520020  9165.780273  6
Proc# 4;  6153.319824  9229.979492  8
Proc# 5;  6196.520020  9294.780273  9
Proc# 6;  6240.520020  9360.780273  5
```

```
Dec 21 21:03 1988  AppB.11.doc Page 2


Proc# 7;  6283.719727  9425.580078  8
Proc# 8;  6328.919922  9493.379883  1
Proc# 9;  6371.319824  9556.979492  13
Proc# 10;  6416.120117  9624.179688  5
Proc# 11;  6459.719727  9689.580078  9

cntbuf: 336, start: 46
hwclock: 19954,  SEQN: 337
# Bad processor ID's = 1  Buffer# 349
# Bad processor ID's = 1  Buffer# 365
# Bad processor ID's = 1  Buffer# 367
# Bad processor ID's = 1  Buffer# 376
# Bad processor ID's = 1  Buffer# 384
# Bad processor ID's = 1  Buffer# 395
# Bad processor ID's = 1  Buffer# 398
iadd=0

Roll: 1,  Xponder: 3  Start: 46
Proc# 0;  6194.120117  9291.179688  4
Proc# 1;  6234.520020  9351.780273  6
Proc# 2;  6275.719727  9413.580078  4
Proc# 3;  6316.919922  9475.379883  4
Proc# 4;  6358.919922  9538.379883  4
Proc# 5;  6370.120117  9555.179688  9
Proc# 6;  -180.679993  0.000000  0
Proc# 7;  6486.120117  9729.179688  4
Proc# 8;  -180.679993  0.000000  0
Proc# 9;  6572.120117  9858.179688  9
Proc# 10;  6616.919922  9925.379883  4
Proc# 11;  6659.319824  9988.979492  5

cntbuf: 414, start: 62
hwclock: 29938,  SEQN: 415
# Bad processor ID's = 1  Buffer# 426
# Bad processor ID's = 1  Buffer# 433
# Bad processor ID's = 1  Buffer# 436
# Bad processor ID's = 1  Buffer# 455
# Bad processor ID's = 1  Buffer# 465
# Bad processor ID's = 2  Buffer# 479

Roll: 1,  Xponder: 4  Start: 62
Proc# 0;  816.119995  1314.099976  49
Proc# 1;  765.720032  1238.500122  56
Proc# 2;  714.920044  1162.300171  59
Proc# 3;  664.119995  1086.099976  41
Proc# 4;  614.119995  1011.099976  51
Proc# 5;  562.920044  934.300049  41
Proc# 6;  512.119995  858.099976  58
Proc# 7;  460.920044  781.300049  57
Proc# 8;  410.119995  705.099976  42
Proc# 9;  360.119995  630.099976  65
Proc# 10;  310.120026  555.100037  35
Proc# 11;  265.320007  487.900024  47
pltsav.scr    1 269 19:08:28.224   5764   5974   6144    342
sh: pltsav.scr: not found
```

```
------------------------------------------------------------------

INPUT FILE:   FLIPproc.new

   Sio Channel Number     1
         1  8.0000     12.000      3.0000      18.000      22.000
         6  16.000     7.0000      21.000      17.000      19.000
        11  20.000     5.0000

------------------------------------------------------------------

INPUT FILE:   f

The first channel contains the Julian day; the second channel
contains the GMT time with the first digit (or two) representing
hours and the second two digits representing minutes;
the third channel contains the slant range from \fIFLIP\fR to the first transpon
the fourth channel contains the slant range from \fIFLIP\fR to the second transp
the last channel contains the slant range from \fIFLIP\fR to the third transpond


   Sio Channel Number     1
         1  254.00     254.00      254.00      254.00      254.00
         6  255.00     255.00      255.00      255.00      255.00
        11  255.00     255.00      255.00      255.00      255.00
        16  255.00     255.00      255.00      255.00      255.00
-----------------
       401  271.00     271.00      271.00      271.00      271.00
       406  271.00     271.00      271.00      271.00      271.00
       411  271.00     271.00      271.00      271.00      271.00
       416  272.00     272.00      272.00      272.00      272.00
       421  272.00     272.00      272.00      272.00
   Sio Channel Number     2
         1  219.00     1700.0      1856.0      2058.C      2306.0
         6  100.00     306.00      500.00      700.00      900.00
        11  1008.0     1100.0      1300.0      1450.0      1614.0
        16  1712.0     1903.0      2013.0      2100.0      2300.0
-----------------
       401  1945.0     2001.0      2016.0      2030.0      2046.0
       406  2102.0     21:8.0      2134.0      2148.0      2201.0
       411  2217.0     2230.0      2245.0      2259.0      2316.0
       416  127.00     201.00      300.00      400.00      500.00
       421  600.00     658.00      905.00      1043.0
   Sio Channel Number     3
         1  4920.8     4906.9      4907.5      4914.1      4909.1
         6  4929.2     4909.6      4914.2      4907.5      4909.7
        11  4915.9     4923.6      4925.0      4916.0      4900.0
        16  4905.8     4904.2      4909.7      4907.6      4922.0
-----------------
       401  4887.1     4885.1      4886.4      4887.1      4881.9
       406  4886.9     4890.3      4896.4      4900.7      4902.9
       411  4902.8     4903.0      4900.0      4902.1      4903.5
       416  4909.1     4909.1      4904.9      4901.4      4896.9
       421  4899.4     4897.1      4902.1      4898.6
   Sio Channel Number     4
```



```
         1  5138.3     5086.7      5090.9      5099.1      5066.2
         6  5056.9     5060.4      5062.5      5065.6      5070.8
        11  5059.4     5036.7      5036.1      5036.1      5071.7
        16  5074.0     5075.0      5066.7      5066.1      5037.5
-----------------
       401  5056.5     5061.9      5058.0      5060.0      5063.0
       406  5059.4     5056.4      5047.6      5043.3      5042.0
       411  5042.8     5039.0      5037.8      5032.7      5031.3
       416  5013.9     5012.4      5014.6      5007.4      5018.8
       421  5024.3     5035.7      5040.2      5036.8
   Sio Channel Number     5
         1  5109.0     5087.0      5085.7      5091.6      5093.3
         6  5093.1     5107.6      5099.5      5105.6      5098.6
        11  5102.9     5119.4      5123.3      5134.4      5106.9
        16  5097.2     5100.0      5103.3      5106.4      5121.5
-----------------
       401  5143.4     5140.1      5141.7      5139.8      5139.0
       406  5140.3     5138.9      5142.0      5142.0      5140.3
       411  5139.8     5143.7      5148.6      5152.8      5151.8
       416  5164.9     5166.1      5169.6      5182.1      5173.3
       421  5163.2     5151.6      5140.8      5150.0

------------------------------------------------------------------

OUTPUT FILE:   out.858

269 19:08:28.224   4909.825   5015.801   5164.826     89.920
  8733.180   8974.979   9291.180   1314.100
  8798.580   9037.979   9351.780   1238.500
  8863.979   9101.580   9413.580   1162.300
  8929.979   9165.780   9475.380   1086.100
  8997.180   9229.979   9538.380   1011.100
  9063.780   9294.780   9555.180    934.300
  9130.979   9360.780      0.000    858.100
  9198.780   9425.580   9729.180    781.300
  9266.580   9493.380      0.000    705.100
  9334.380   9556.979   9858.180    630.100
  9404.580   9624.180   9925.380    555.100
  9471.180   9689.580   9988.979    487.900
```

APPENDIX C. Array and *FLIP* Localization Program Description.

FLPNAV

Most of the code for this program is involved in reading the inputs. There are some minor system subroutines called which interpret ascii input files. Only subroutines which pertain to the array navigation problem will be described here.

Main Program	Subroutines	
	Level 1	Level 2
FLPNAV	linint xcor fix3	

FLPNAV: Locates the array receiver positions in a least squares sense using parameters entered by the user in a command file. User parameters define the transponder locations, slant range input data file name, output file name, sound speed profile, interpolation method for the *FLIP* slant ranges and a method for constraining deviations in slant range with time. The set of measurements taken at any one particular time are referred to as a "fix". Some parameters may be input from the command file or from separate existing files e.g. *dvfile filename* inputs depth vs. sound speed values from the named file. Parameters which are not specifically declared in the command file are preset to the value indicated below. Some of the parameters defined below were used for test purposes only.

The information required by this program is:

1) Fixed x, y, and z positions for each of 3 transponders.
2) Initial x y positions for each receiver and *FLIP*.
3) Fixed depth of each receiver and *FLIP*.
4) The slant range of each receiver and *FLIP* to each transponder. The array receiver slant ranges must be the distance from *FLIP* to transponder to array receiver.
5) Sound speed profile.

The navigation procedure is:

1) Discard noisy slant ranges using the parameters THRESD and THRES. An entire fix is thrown away if more than 7 of the slant ranges for any one transponder are zero (this is to find the fixes that occur when the navigation is turned off - usually around the hour when *FLIP* is navigated).
2) Adjust the receiver depths and slant range measurements using the harmonic mean based on the sound speed profile measured by a CTD.
3) Interpolate for a new *FLIP* slant range set if files are available; subtract the *FLIP* slant range from the receiver slant range leaving the range from transponder to receiver.
4) Calculate a horizontal range for *FLIP* and each array receiver
5) Iterate the *FLIP* and receiver positions minimizing the rms error in a least squares sense.

Input Parameter Definitions:

ifile - The file name of the file containing the slant ranges and depths of *FLIP* and the array receivers whose positions are to be iterated. A *FLIP* slant range is the path from *FLIP* to the transponder, a receiver slant range is the path from *FLIP* to the transponder to the receiver.
Preset = rnglog.dat   e.g. ifile /localdata/ranges/data

ofile - The output file name. When given, the output xyz information will be written to file *ofile* instead of standard out.
Preset = ' '   e.g. ofile output

trlocs - A list of the transponder locations, given as x, y, z, and variance for each transponder.
Preset = none e.g. trlocs 1000 1000 2000 1.0 3000 1000 2100 2.0 2000 2000 1500 1.0

depths - A list of depths of all the receivers, replacing the receiver depths in *ifile*. All depths including *FLIP*'s must be given if any are given. DEPTHS is ignored unless given.
Preset = 0   e.g. 90 300 375 450 525 600 675 750 825 900 975 ....

nsta - The number of fixes in *ifile*.
Preset = 1

stainc - The increment between fixes in *ifile* to interate. The first fix in *ifile* is always the first. e.g. *stainc* 2 skips every other fix.
Preset = 1.   e.g stainc 10

n2ave - The number of consecutive fixes to average before trying to calculate each location. The average is performed by doing a running average of one-way slant ranges (transponder to receiver) of *n2ave* fixes. Noisy fixes (where more than 1/2 of the ranges are "noisy") are not included in the average. *n2ave* may not exceed 10.
Preset = 1   e.g. n2ave 10

alter - The alternatives for noisy slant range data. Slant ranges more than *thres* different from the slant range for the same receiver/ transponder of the previous day/time will be "noisy".

=0, No data are discarded as being "noisy".

=1, A single noisy slant range out of the three slant ranges for a single receiver will cause the data for that receiver to be discarded. The output file will not contain a position for the receiver for that day/time. The file will still contain the total number of receivers, but some receiver positions will be duplicated (this allows plot programs to just plot the same receiver on top of itself).

=2, The noisy slant range will be replaced by the slant range for the same receiver and transponder from the previous day/time in the input file.

=3, The "noisy" slant ranges will be replaced by interpolating the slant range to the same transponder from adjacent receivers. A noisy receiver on either end of the array will receive the slant range from the previous fix. Likewise, if two adjacent receivers have noisy slant ranges, the first one will receive the slant range for that receiver from the previous day/time.

=4, Noisy slant ranges are calculated by extrapolation or interpolation of adjacent non-zero slant ranges.

Preset = 3

thres - The threshold slant range in meters used in determining noisy slant ranges.
Preset = 100.   e.g. thres 50

thresd - The threshold depth difference in meters used in determining "noisy" depths. A depth is considered noisy if it is more than *THRESD* away from the depth of the same receiver on the previous day/time.

Preset = 5

trfile - A file containing the transponder locations (x, y, z) in the format written by program XPMAIN with an additional column specifing the variance.
Preset = none e.g. trfile trs

dvfile - A file containing the depth-sound speed pairs as described below.

vdp - A list of sound speed depth-pairs to use in making sound speed corrections. The sound speed correction is made by first converting every depth and slant range to time using a constant sound speed of *oldcv*. The speeds in *vdp* are assumed to be the speed of sound at the depth given. The speed used between specified depths is the "interval" speed which is the average speed of the two adjacent depths.

dvp - A list of depth-speed pairs. This is identical to *vdp*, except that the order of the pairs is depth followed by speed.
Preset = none

oldcv - The constant speed used in obtaining the slant ranges.
Preset = 1500. e.g. oldcv 1450

calctd - Assuming that the array is vertical and directly below *FLIP*, data from two array receivers may be used to estimate the transponder depths. Enter the two array receiver numbers for the calculation. Test mode only.
Preset = 0 e.g. calctd 3 4

lpfile - The pathname of an output file containing the slant ranges in the format expected by the looping program XPMAIN. The file contains the slant ranges from the transponder to each receiver.
Preset = none e.g. lpfile slr.loop

erfile- The pathname of a file in which data error messages will be written. No data error message will be written unless *erfile* is given.
Preset = none e.g. erfile oops

fudge - A factor to test *FLIP*'s slant ranges. Test mode only.
Preset = 0. e.g. fudge 10.

Subroutines:

linint: Interopolates the *FLIP* slant ranges linearly.

xcor: The harmonic mean of the sound speed profile is used in correcting the slant ranges and depths which were calculated assuming a constant sound speed of 1500 m/s.

trgss: Not used for the September 87 data. Calls Fix2 to calculate two possible 'fix' positions. The position closest to the third transponder position is selected.

fix2: Nor used for the September 87 data. Calculates the two positions defined by the intersection of the slant ranges from the first two transponders. See Appendix C.

fix3: Adjusts the of first the *FLIP* position and then each array position using a single fix to obtain a least squares convergence. Outputs the adjusted xy position and the rms error for each call.

```
-----------------------------------------

SAMPLE RUN of flpnav:

Script started on Wed Dec 21 19:36:11 1988
C51>ff.scr
  0   errors in the job.
  trloc   646.0000    4128.000    4564.000
  trloc   2593.000    872.0000    4566.000
  trloc   4383.000    4736.000    4573.000
   269.7975    4909.189    5015.365    5164.743
   269.7983    4909.342    5015.155    5164.839
   269.7990    4909.495    5014.944    5164.935
   269.7998    4909.648    5014.733    5165.030
   269.8006    4909.802    5014.522    5165.126
   269.8013    4909.955    5014.312    5165.222
   269.8021    4910.108    5014.101    5165.318
   269.8029    4910.262    5013.890    5165.414
   269.8036    4910.415    5013.679    5165.509
   269.8044    4910.568    5013.469    5165.605
   269.8051    4910.722    5013.258    5165.701
   269.8059    4910.875    5013.047    5165.797
   269.8066    4911.028    5012.836    5165.893
   269.8074    4911.181    5012.626    5165.988
   269.8082    4911.334    5012.415    5166.084
   269.8089    4911.488    5012.204    5166.180
   269.8097    4911.641    5011.993    5166.276
   269.8104    4911.794    5011.783    5166.372
   269.8112    4911.948    5011.572    5166.467
   269.8120    4912.101    5011.361    5166.563
   269.8127    4912.254    5011.150    5166.659

-----------------------------------------

INPUT COMMAND FILE:  ff.scr

 Nflpnav<<eof
  nsta 11111 thresd 7
  ifile out.858t
  ofile xyz.tst.858t
  thres 15
  alter 4
  redf red.flip
  greenf green.flip
  bluef blue.flip
trfile trs2
dvp
 0        1510.0     29.781   1502.6     39.707   1500.7     59.558   1499.1
69.483    1498.2     79.407   1497.4     89.331   1494.9     99.254   1495.6
119.10    1492.7     138.94   1487.5     158.78   1486.0     178.62   1485.3
198.46    1484.9     218.30   1484.1     238.13   1483.7     277.79   1482.2
297.62    1481.7     330.00   1481.2     350.00   1481.5     376.91   1481.3
396.71    1480.8     416.54   1481.1     426.45   1481.2     456.17   1480.6
495.79    1480.2     505.69   1479.4     525.50   1479.3     555.21   1479.2
```



```
575.01    1478.8     584.91   1478.7     604.70   1478.8     614.60   1479.2
644.30    1479.3     664.09   1478.9     683.88   1478.8     713.56   1478.9
861.91    1479.9     881.68   1480.0     901.45   1480.3     940.99   1480.5
980.51    1480.8     1000.3   1481.0     1049.7   1481.3     1108.9   1481.6
1207.6    1482.3     1306.3   1483.3     1405.0   1484.3     1503.5   1485.3
1749.8    1487.9     2005.6   1491.2     2506.5   1498.7     3006.2   1506.8
3504.8    1515.2     4007.1   1524.0     4906.5   1540.4
end
eof

------------------------------------------------------------------------

INPUT FILE:  trs2

 24-Jul-88 12:52:32 looping, RMS =   0.74 based on  345 fixes from flip.indat
           646.    4128.  4564.  1.0
          2593.     872.  4566.  1.0       3793.7
          4383.    4736.  4573.  1.0       3785.9  4258.1

------------------------------------------------------------------------

INPUT FILE:  red.flip   green.flip   blue.flip

red.flip, green.flip and blue.flip contain the data from flip.indat reformatted

time slntrng
139 4920
1020 4906
1136 4907
1258 4914
1386 4917
1500 4929
1626 4909
.... ....

-----------------
```

APPENDIX D. Initial Position Calculation.

To calculate an array X-Y position (Figure D.1), the horizontal range from FLIP or the array to any transponder is given by

$$Hproj\,1 = S^2 - (D_{T1} - D_{FLIP})^2$$

Hproj1 is the horizontal range from transponder 1 to FLIP,
S is the one way slant range from FLIP to transponder 1,
D_{T1} is the depth of transponder 1 and
D_{FLIP} is the depth of the transmitter mounted on the bottom of FLIP.

Two X-Y positions are calculated from the intersection of the arcs defined by the horizontal projections:
X_{T1}, Y_{T1} = the X-Y position of transponder 1;
X_{T2}, Y_{T2} = the X-Y position of transponder 2;
X_{12} and Y_{12} are the differences between the X-Y transponder positions ($X_{12} = X_{T1} - X_{T2}$);
Bline = the transponder baseline range = $(X_{12}^2 + Y_{12}^2)^{1/2}$;
Hproj1 = transponder 1 to fix range;
Hproj2 = transponder 2 to fix range;
P1 = projection of fix onto baseline point to transponder 1
P2 = projection of fix onto baseline point to transponder 2
C = perpendicular distance from fix to baseline
X_{PJ} = X-coordinate of the projection of the fix onto the baseline
Y_{PJ} = Y-coordinate of the projection of the fix onto the baseline
X1,Y1 = X-Y position of the first fix
X2,Y2 = X-Y position of the second fix

$$Bline = P\,1 + P\,2$$

$$C^2 = Hproj\,1^2 - P\,1^2 = Hproj\,2^2 - P\,2^2 = Hproj\,2^2 - (Bline - P\,1)^2$$

$$Hproj\,1^2 - P\,1^2 = Hproj\,2^2 - Bline^2 + 2P\,1\,Bline - P\,1^2$$

$$P\,1 = \frac{Hproj\,1^2 - Hproj\,2^2 + Bline^2}{2Bline}$$

$$\cos\theta = \frac{X_{12}}{Bline};\ \sin\theta = \frac{Y_{12}}{Bline}$$

$$X_{PJ} = X_{T1} + P\,1\cos\theta;\ Y_{PJ} = Y_{T1} + P\,1\sin\theta$$

$$X\,1 = X_{PJ} - C\sin\theta;\ Y\,1 = Y_{PJ} + C\cos\theta$$

$$X\,2 = X_{PJ} + C\sin\theta;\ Y\,2 = Y_{PJ} - C\cos\theta$$

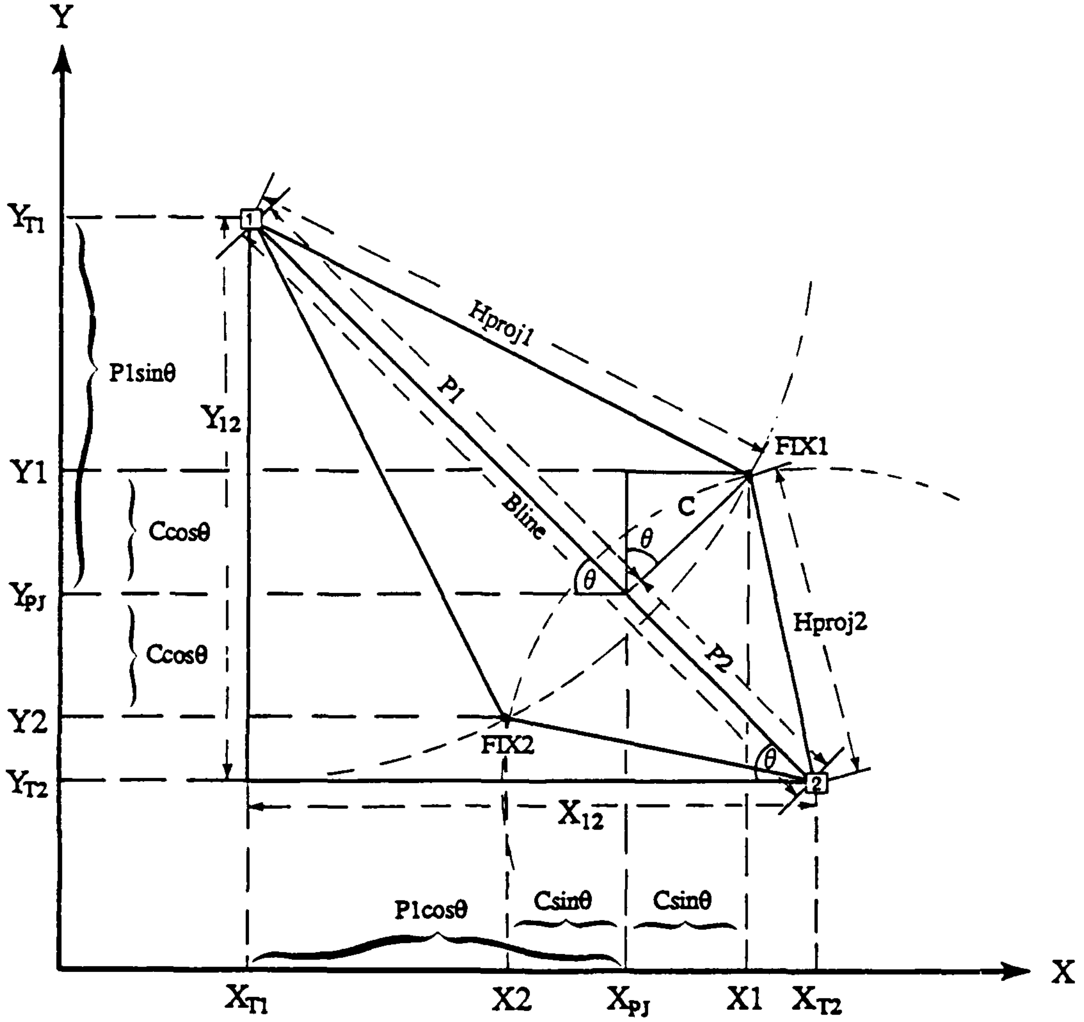

Figure D.1 Array Position Calculation. The parameters required to calculate the lateral position of an array element. The points at FIX1 and FIX2 give the intersection of the horizontal projection arcs for two transponders. A third transponder range is used to choose between these two positions.

APPENDIX E. Software Listings.

The software discussed in this report is listed below. The listings contain the main program, subroutines, makefile and include files for each program discussed.

```
#
# make file for deep transponder navigation looping program
#
FILES= xpmain.f xprint.f basln2.f xploop.f xpfil.f xpurge.f \
        xpcmld.f xpread.f xpset.f xpinpt.f tmdate.f atof.c \
        xprin2.f xxcor.f

OBJECTS= xpmain.o xprint.o basln2.o xploop.o xpfil.o xpurge.o \
        xpcmld.o xpread.o xpset.o xpinpt.o tmdate.o atof.o \
        xprin2.o xxcor.o

LIBS=/localdata/sun/LIB/libdeeptow.a /localdata/sun/LIB/SUBS/newsubs.a

FFLAGS= -g -ffpa

all:    xpmain xpload

xpmain: ${OBJECTS}
        f77 ${FFLAGS} ${OBJECTS} ${LIBS} -o Txpmain

xpload: xpload.o xpcom.h xpcmld.o xpset.o atof.o
        f77 ${FFLAGS} xpload.o xpcmld.o xpset.o atof.o ${LIBS} -o xpload

print:
        @pr xpload.f
        @pr ${FILES}
        @pr xpcom.h Makefile

.f.o:
        f77 ${FFLAGS} -c $*.f
#
# dependances
#
xpmain.o:       xpcom.h
xprint.o:       xpcom.h
xploop.o:       xpcom.h
xpfil.o:        xpcom.h
xpurge.o:       xpcom.h
xpcmld.o:       xpcom.h
xpread.o:       xpcom.h
xpset.o:        xpcom.h
xpinpt.o:       xpcom.h
atof.o:         atof.c
xprin2.o:       xpcom.h
```

```
      SUBROUTINE xxcor ( xold, xdepth, vold, vd, nvdp, xnew, z )
c     XCOR corrects a distance (xold) for an incorrect sound speed (vold) and
c  applies a new one (vdp).
c     A travel time is calculated by assuming xold was computed with a
c  constant sound speed (vold).  The new distance is calculated using a
c  summation of interval speeds.  The structure of VDP must be such
c  that a constant speed is used for the interval between the depths on either
c  side in the vdp array.  This approach assumes that an XBT or CTD was used to
c  measure the instantaneous speed for each depth and that the speed is
c  constant for the interval between depth samples.  The speed used is the
c  average speed, assuming that the interval speed is the average between
c  the 2 instantaneous speeds. We know the ray is not vertical.  The initial
c  angle can be measured by using the known depth (z) and the slant range (xold)
c  If we ASSUME that the angle stays the same, we can calculate the length of
c  time the c  ray spends in each layer.  Snell's law or ray bending is ignored.
c  The vdp array assumes that there is a constant speed from the surface to the
c
c  ARGUMENTS:
c  xold    - The distance to be modified.
c  xdepth  - The depth of the object where xold starts (the starting depth
c            of the ray).
c  vold    - The constant speed used in obtaining xold.
c  vdp     - An array of speed-depth pairs to use in calculating XNEW
c  nvdps   - The number of elements in the vdp array.  There must be
c            nvdps/2 pairs in the vdp array.
c  xnew    - The new distance calculated.  XNEW is returned by XCOR.  XNEW may
c            be the same as XOLD.
c  z       - The depth of the final point of the ray.
c
c
      PARAMETER (nvdps = 56*2 )
      DIMENSION vdp(nvdps)
      DATA vdp/1510., 0., 1502.6, 29.781, 1500.7, 39.707,
     *      1499.1, 59.558, 1498.2, 69.483, 1497.4, 79.407,
     *      1494.9, 89.331, 1495.6, 99.254, 1492.7, 119.1,
     *      1487.5, 138.94, 1486., 158.78, 1485.3, 178.62,
     *      1484.9, 198.46, 1484.1, 218.3, 1483.7, 238.13,
     *      1482.2, 277.79, 1481.7, 297.62, 1481.2, 330.,
     *      1481.5, 350., 1481.3, 376.91, 1481.8, 396.71,
     *      1481.1, 416.54, 1481.2, 426.45, 1480.6, 456.17,
     *      1480.2, 495.79, 1497.4, 505.69, 1479.3, 525.5,
     *      1479.2, 555.21, 1478.8, 575.01, 1478.7, 584.91,
     *      1478.8, 604.7, 1479.2, 614.6, 1479.3, 634.4,
     *      1479.3, 644.3, 1478.9, 664.09, 1478.8, 683.88,
     *      1478.9, 713.56, 1479.9, 861.91, 1480., 881.68,
     *      1480.3, 901.45, 1480.5, 940.99, 1480.8, 980.51,
     *      1481., 1000.3, 1481.3, 1049.7, 1481.6, 1108.9,
     *      1482.3, 1207.6, 1483.3, 1306.3, 1484.3, 1405.,
     *      1485.3, 1503.5, 1487.9, 1749.8, 1491.2, 2005.6,
     *      1498.7, 2506.5, 1506.8, 3006.2, 1515.2, 3504.8,
     *      1524., 4007.1, 1540.4, 4906.5 /
c
c
      theta = (z-xdepth)/xold
      IF( theta .LT. .0001 ) THEN
          rtheta = 1.
```



```
      ELSE
          rtheta = 1. / theta
      ENDIF
      t = xold / vold                    ! find the original one-way travel time
      x = 0.
      tleft = t
      IF( nvdps .EQ. 2 ) THEN      ! is the new speed a constant speed?
          xnew = t * vdp(1)
          RETURN
      ENDIF
      IF( t*vdp(1) .LE. vdp(2) ) THEN        ! is it in the first interval?
          xnew = t * vdp(1)
          RETURN
      ENDIF
c
      idone = 0                         ! a flag indicating completion when = 1
c****
c****  find the first vdp that is after xdepth
c****
      DO 50 k = 1, nvdps, 2
         i = k
         IF( xdepth.LT. vdp(k+1) ) THEN
             deltax = vdp(i+1)-xdepth                   ! do the first one
             x = deltax * rtheta
             tleft = tleft - x/vdp(i)
             GOTO 190
      ENDIF
   50 CONTINUE
      xnew = t * vdp(nvdps-1)                 ! must be after the last depth!
      RETURN
c****
  100 CONTINUE
      v = ( vdp(i) + vdp(i-2) ) / 2.      ! the interval speed is the average
      deltad = vdp(i+1) - vdp(i-1)             ! the length of the interval
      deltax = deltad * rtheta            ! the total distance in the interval
      deltat = deltax / v                  ! the time spent in the interval
      IF( tleft - deltat .GT. 0. ) THEN
CC***! does the distance end in this interval?
         x = x + deltax                            ! add in this interval
         tleft = tleft - deltat       !subtract the time spent in the interval
      ELSE
         x = x + tleft * v
C****    ! use the interval speed even if it terminated within the interval
         GOTO 200
      ENDIF
  190 i = i + 2
      IF( i .LT. nvdps ) GOTO 100
      x = x + tleft * vdp(i-2)
C****                  ! use the last speed all the way out (extrapolate)
  200 CONTINUE
      xnew = x
      RETURN
      END
```

Dec 16 09:39 1988 xpmain.f Page 1

```
C
C       This program calculates the set of transponder positions most
C       nearly consistant with a set of concurrently observed ranges
C       from the transponders.
C
C       There are five files associated with this program.
C       xpcom  is the input common file (contains transponder positions from xpl
C       namfil is the input 12KHz position (x,y,slant) file
C       ifile  is the output transponder position file
C       efile  is the output error/info file
C       outfil is the output data file which can be used as an interative input
C
C
C
        LOGICAL          BEGUN
C
        IMPLICIT REAL*4 (A-H, O-Z)
C
        CHARACTER*4      LABEL(15)
        CHARACTER*9      DATSTR
        CHARACTER*8      TIMSTR
        CHARACTER*80     NAMFIL,IFILE,EFILE
C
        INTEGER          GTNUM
C
        include 'xpcom.h'
C
        REAL*4           TRX(15),TRY(15),TRZ(15)
        DIMENSION        TX(16), TY(16)
C                                         !used to remember initial xpndr psns
        DIMENSION        BSLTRU(15), BSLCLC(15), DIFF(15)
C
        DATA NPOMX/1500/, RDMIN /.5/, RELERR /.001/
        DATA             LABEL /15*' '/
        DATA             NULL /6/
C
        IFLAG = 0
        BEGUN = .FALSE.
        LDFLG=0
C                                         !ensures data is read from disk into
        CALL XPCMLD
C                                         ! XPCOM by the xponder version of CMLD
        CALL XPINPT(NAMFIL,IFILE,EFILE,ITIMES)
C                                         !get input parameters
        DUMMY(1) = 2.0
        IF(GTNUM('RMS error factor for fix rejection (default=2) : ',
     *     AAA,1) .gt. 0) DUMMY(1) = AAA
        DUMMY(2) = 0.0
        IF(GTNUM('Do you want to correct the slant ranges for sound
     *     speed? (NO=0,YES=1) : ',AAA,1) .gt. 0) DUMMY(2) = AAA
C
        DO 110 NTR = 1,NTRS
C                                         !Save "true" positions
         IF (DUMMY(2) .eq. 1.0) then
          call xxcor(TRPN(NTR,3),0.0,1500.,0.0,0,TRPN(NTR,3),
     *        TRPN(NTR,3))
```

Dec 16 09:39 1988 xpmain.f Page 2

```
         ENDIF
          TX(NTR) = TRPN(NTR,1)
          TY(NTR) = TRPN(NTR,2)
  110   CONTINUE
C
        ITIMES = 1
        DO 4000 ITIME = 1,ITIMES
C                                         !For each "realization"...
C
        WRITE(LSCRN,124) NAMFIL
  124   FORMAT(' Navigation file is ',A)
C
C
        NPOS=0
C                                         !initialize number of positions in store
        OLDERR=10000.0
C                                         !initialize error from previous loop
        NOMORE=0
C                       !flag that data file has not been exhausted
C
2000    CONTINUE
        CALL XPREAD(NAMFIL,OLDERR)
C                                         !fill CRANS
        WRITE(LERR,2100) NPOS
2100    FORMAT(//,XI5,' POSITIONS IN STORE')
2400    CONTINUE
        CALL XPLOOP
C                                         ! to do the looping and adjusting.
        IF(RMSERR * SQRT(FLOAT(NPOS)) .LT. 1.0) GO TO 2500
        IF((OLDERR-RMSERR)/OLDERR .LT. RELERR) GO TO 2500
        IF(RMSERR .LT. .75) GO TO 2500
C       CALL SCCA(IFLAG)
C       IF(IFLAG .NE. 0) GO TO 2500
C                                         !FORCED QUIT BY ^C^C
        OLDERR=RMSERR
        CALL XPURGE
        WRITE(LERR,2503)
        CALL BASLN2(NULL,NTRS,TRPN(1,1),TRPN(1,2),BSLCLC,NCALC,IXPCD)
C                                         !deriv
        DO 1810 I=1,NTRS
            TRX(I) = TRPN(I,1)
            TRY(I) = TRPN(I,2)
            TRZ(I) = TRPN(I,3)
            LABEL(I) = IXPCD(I)
 1810       CONTINUE
C
        CALL TMDATE(DATSTR,TIMSTR)
        WRITE(LERR,811) DATSTR,TIMSTR,RMSERR,NPOS,NAMFIL
        WRITE(LERR,821) LABEL(1),TRX(1),TRY(1),TRZ(1),
     *            (IXPCD(I),I=1,NTRS-1)
        N = 0
        DO 1830 I=2,NTRS
            WRITE(LERR,822) LABEL(I),TRX(I),TRY(I),
     *            TRZ(I),(BSLCLC(N+J),J=1,I-1)
            N = N + I -1
 1830       CONTINUE
```

```
      GO TO 2000
C
 2500 CONTINUE
C                                        !to print out the results.
      WRITE(LERR,2501)
 2501 FORMAT(' Initial transponder baseline lengths...',/)
      CALL BASLN2(LERR,NTRS,TX,TY,BSLTRU,NTRU,IXPCD)
C                                        !initial lengths
      WRITE(LERR,2503)
 2503 FORMAT(' Baseline lengths derived by looping...',/)
      CALL BASLN2(LERR,NTRS,TRPN(1,1),TRPN(1,2),BSLCLC,NCALC,IXPCD)
C                                        !derived
C
C   ...Generate a table of baselines differences and form an error estimate
C      for all baselines.
C
      N = 0
      DIFSQR = 0.00
      SUMDIF = 0.00
      DO 3000, I = 1,NTRU
          IF(BSLTRU(I) .EQ. 0.00 .OR. BSLCLC(I) .EQ. 0.00)
     *    GO TO 3000
              N = N + 1
              DELTA =  BSLCLC(I) - BSLTRU(I)
C             TYPE *,'TRU,CALC,DIFF,I=',
C    *            BSLTRU(I),BSLCLC(I),DELTA,I
              DIFF(I) = DELTA
 3000     CONTINUE
C
C
      WRITE(LERR,3001)
 3001 FORMAT(' Differences between initial and "looped" baselines...'/)
C
CD    N = 0
CD    DO 3150 I = 1,NTRS -1
CD        TYPE *,(N+J,J=1,I)
CD        N = N + I
CD3150 CONTINUE
C
      N = 0
      WRITE(LERR,3102) (IXPCD(I),I=1,NTRS-1)
      DO 3200 I = 1,NTRS -1
          WRITE(LERR,3101) IXPCD(I+1),(DIFF(N + J),J=1,I)
          N = N + I
 3101     FORMAT(1X,A2,6(1PG16.5))
 3102 FORMAT(3X,6(7X,A2,7X))
 3200 CONTINUE
      CALL XPRINT
 4000 CONTINUE
C
      DO 810 I=1,NTRS
        TRX(I) = TRPN(I,1)
        TRY(I) = TRPN(I,2)
        TRZ(I) = TRPN(I,3)
        LABEL(I) = IXPCD(I)
  810   CONTINUE
```



```
C
      OPEN(UNIT=18,STATUS='UNKNOWN',
     *                        FILE=IFILE,FORM='FORMATTED')
      CALL TMDATE(DATSTR,TIMSTR)
      WRITE(18,811) DATSTR,TIMSTR,RMSERR,NPOS,NAMFIL
  811 FORMAT(1X,A,1X,A,' looping, RMS ='F7.2,' based on ',I4,
     *      ' fixes from ',A)
      WRITE(18,821) LABEL(1),TRX(1),TRY(1),TRZ(1),
     *              (IXPCD(I),I=1,NTRS-1)
  821     FORMAT(X,A,4X,2(F7.0,1X),F6.0,3X,7(6X,A2))
      N = 0
      DO 830 I=2,NTRS
          WRITE(18,822) LABEL(I),TRX(I),TRY(I),TRZ(I),
     *                  (BSLCLC(N+J),J=1,I-1)
          N = N + I -1
  822     FORMAT(X,A,4X,2(F7.0,1X),F6.0,5X,7(1X,F7.1))
  830     CONTINUE
      CLOSE(UNIT=18)
C     CLOSE(UNIT=LERR)
C
      CALL XPRIN2
C
      CALL EXIT(0)
      END
```

```
      SUBROUTINE XPCMLD
C             75-APR-17
C     XPCMLD,LOADS THE TRANSPONDER COMMON AREA FROM THE XPCOM
C     DISK FILE ON DISK,WHEN LDFLG,LOAD FLAG,IS NEGATIVE OR ZERO.
C     IT WRITES THE DATA ONTO DISK WHEN LDFLG IS POSITIVE.
C     NOTE THAT EACH CALL TO LDFLG CHANGES LDFLG
C     TO INDICATE THE CURRENT STATUS OF THE SYSTEM.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      integer*4 LUTRC
C
      include 'xpcom.h'
C
      LOGICAL L1
C
      DATA LUTRC/19/
C             LUTRC,LOGICAL UNIT NUMBER OF TRANSPONDER COMMON FILE.
      PRINT *, 'LUTRC = ', LUTRC
      INQUIRE(FILE='XPCOM.DAT',EXIST=L1)
      IF (.NOT.L1) GO TO 100
      OPEN (UNIT=LUTRC,FILE='XPCOM.DAT',STATUS='OLD',
     *      FORM='UNFORMATTED',ERR=100)
      PRINT *,'XPCOM.DAT FOUND, OLD FILE OPENED...'
      GO TO 150
 100          CONTINUE
              PRINT *,'XPCOM.DAT NOT FOUND, NEW FILE OPENED'
              OPEN (UNIT=LUTRC,FILE='XPCOM.DAT',FORM='UNFORMATTED',
     *                STATUS='NEW')
              LDFLG = 1
              CALL XPSET
 150  CONTINUE
CD    PRINT *,'LDFLG =',LDFLG
      IF(LDFLG) 1000,1000,2000
1000  CONTINUE
      LDFLG=1
      READ (LUTRC,err=3000,end=3000) SPARE,TRPN,
     *        SCALE,XL,XR,YB,YT,XO,YO,
     *        SURFV,DEEPV,DUMMY,ISPAR,LFINL,NTRS,
     *        LINTER,LERR,LSPAR,LDFLG,IXPCD,
     *        LSCRN,LNEW
      GO TO 3000
2000  CONTINUE
      WRITE (LUTRC) SPARE,TRPN,SCALE,XL,XR,YB,YT,XO,YO,
     *        SURFV,DEEPV,DUMMY,ISPAR,LFINL,NTRS,
     *        LINTER,LERR,LSPAR,LDFLG,IXPCD,
     *        LSCRN,LNEW
      LDFLG=-1
C
3000  CONTINUE
CDEBUG START
CD    PRINT *,'ISPAR=',ISPAR
CD    PRINT *,'SPARE=',SPARE
CD    PRINT *,'NTRS,TRPN()=',NTRS
CD    DO 3010 I=1,NTRS
CD    PRINT *,(TRPN(I,J),J=1,4)
```



```
CD3010  CONTINUE
CD      PRINT *,'SCALE,XL,XR,YB,YT,XO,YO,LDFLG=',
CD     *                SCALE,XL,XR,YB,YT,XO,YO,LDFLG
        PRINT *,'LFINL,LINTER,LERR,LSPAR=',LFINL,LINTER,LERR,LSPAR
        PRINT *,'LSCRN,LNEW=',LSCRN,LNEW
CD      PRINT 3011,(IXPCD(I),I=1,NTRS)
CD3011  FORMAT(15(3X,A2))
CD      PRINT *,'SURFV,DEEPV=',SURFV,DEEPV
CD      PRINT *,'DUMMY=',DUMMY
CDEBUG END
        CLOSE(LUTRC)
        RETURN
        END
```

```
      SUBROUTINE XPSET
C
C           FILE XPSET.FOR
C
C
C     GLOSSARY......
C
C     IXPCD   TRANSPONDER NAMES IN ASCII CODE.
C             G1      GREEN ONE (10.0 KHZ TRANSPONDER).
C             G2      GREEN TWO (10.0 KHZ TRANSPONDER).
C             ..      .......
C             G5      GREEN FIVE.
C             R1      RED ONE (10.5 KHZ TRANSPONDER).
C             R2      RED TWO.
C             ..      ......
C             R5      RED FIVE.
C             B1      BLUE ONE (11.0 KHZ TRANSPONDER).
C             ETC.    ETC.ETC. UP TO
C             B5      BLUE FIVE.
C     NTRS    NUMBER OF TRANSPONDERS IN USE.MAXIMUM OF 15.
C     TRPN    X,Y,Z COORDINATES OF TRANSPONDERS.
C     FOR TRPN (NPO,J)
C             NPO-TRANSPONDER NUMBER.
C             J=1 X POSITION.
C             J=2 Y POSITION.
C             J=3 Z POSITION.
C             J=4 TURNAROUND DELAY
C
C     LFINL LOGICAL UNIT NUMBER FOR THE FINAL LISTING OF
C             NAVIGATION DATA AT THE END OF EACH FIX.
C     LINTER LOGICAL UNIT NUMBER OF DEVICE USED FOR LISTING
C             OF DATA DURING COMPUTATIONS.
C     LERR LOGICAL UNIT NUMBER OF DEVICE USED FOR LISTING
C             OF ERROR MESSAGES.
C     LSCRN  LOGICAL UNIT FOR SCREEN OUTPUT
C     LSPAR UNUSED AT PRESENT.
C     SCALE   SCALE IN USERS UNITS PER INCH OF PLOTTER.
C             NOTE DIFFERENC IN CONVENTION BETWEEN PDP-11 AND 1800.
C     XL,XR   LEFT AND RIGHT HAND LIMITS OF PLOTTER DISPLAY IN USER UNITS.
C     YB,YT   BOTTOM AND TOP LIMITS OF Y-AXIS OF PLOTTER IN USER UNITS.
C     XO,YO   CO-ORDINATES SPECIFYING POSITION WHERE PLOTTER
C             PEN IS TO BE RETURNDE TO AFTER PLOTTING.(USER UNITS).
C     LDFLG   LOAD FLAG,IS POSITIVE WHEN THE MOST CURRENT TRANSPONDER
C             DATA IS IN CORE,NEGATIVE OR ZERO WHEN THE PRIMARY SET OF DATA
C             RESIDES ON DISK. SEE CMLD FOR DETAILS.
C     DEEPV   ASSUMED VELOCITY OF SOUND AT OPERATING DEPTH.
C     SURFV   ASSUMED HARMONIC MEAN OF SOUND VELOCITY FROM SURFACE TO DEPTH.
C             SEE TRSLA FOR DETAILS.
C             ISPAR(1) IS RESERVED FOR XPCMLD TO USE FOR NPOMX.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      include 'xpcom.h'
C
C     CLEAR OUT COMMON AREA BY FILLING WITH ZEROES.
      DO 2000 ITRAN=1,15
```



```
      SPARE(ITRAN) = 0.
      ISPAR(ITRAN) = 0
      IXPCD(ITRAN) = ' '
      DO 1800 INDEX=1,4
1800  TRPN(ITRAN,INDEX)=0.0
2000  CONTINUE
      NTRS=0
      XL=-15000.0
      XR=15000.0
      YB=-15000.0
      YT=15000.0
      XO=15000.0
      YO=15000.0
      SCALE=1000.0
C
C             LOAD LOGICAL UNIT NUMBERS.
C             TEMPORARY SET UP FOR BEACH USE WITH ALL OUTPUT ON LP:
      LSCRN=6
      LFINL=6
      LNEW=37
      LINTER=6
      LERR=6
      LSPAR=8
C     SET SOUNDVELOCITIES TO NOMINAL VALUES.
      DEEPV=1500.0
      SURFV=1500.0
      NPOS=0
      END
```

```
      SUBROUTINE XPLOOP
C
C     XPLOOP loops the positions of the transponders and the fishes
C     to get better agreement.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      CHARACTER*6     NAME
C                                      !for id on output.
C
      include 'xpcom.h'
C
      DATA MXLOOP/30/
      DATA RDMIN /0.00350/
C                                      !minimum reduction of error to
C
C                                      ! continue looping.
      ERLST=100000000.00
C                                      !initial value for previous error.
C
      DO 4000 LOOP=1,MXLOOP
      DO 3100 NPO=1,NPOS
C                                      !loop each fish position.
      OUT=TIM(NPO)/1000.0
C                                      !bring time to hours.
      WRITE(NAME,1500) OUT
1500  FORMAT(F6.3)
3100  CALL XPFIL(TRPN(1,1),TRPN(1,2),CRANS(1,NPO),NTRS,1,NAME,
     *           PSNS(NPO,1),PSNS(NPO,2),PSNS(NPO,3))
C
CD     PRINT *,'TIME,X,Y,ERR=',TIM(NPO)/1000.,PSNS(NPO,1),PSNS(NPO,2),
CD    *                        PSNS(NPO,3)
      ERSUM=0.0
      NAME(3:6)='    '
      NMOVED=0
      DO 3200 NTR=NTRS,1,-1
      print *,ntr
          IF(IMOVE(NTR) .LE. 0) GO TO 3200
C                                      !HOLD or IGNORE ?
              NMOVED=NMOVED+1
C                                      ! no
              NAME=IXPCD(NTR)
              CALL XPFIL(PSNS(1,1),PSNS(1,2),CRANS(NTR,1),NPOS,6,
     *          NAME,TRPN(NTR,1),TRPN(NTR,2),ERR)
              ERSUM=ERSUM+ERR*ERR
3200      CONTINUE
C
      RMSERR=SQRT(ERSUM/NMOVED)
      WRITE(LERR,3400) LOOP,RMSERR
3400  FORMAT(' LOOP NO  ',I4,' HAS AN ERROR OF ',G11.5)
      IF((ERLST-RMSERR)/ERLST-RDMIN) 4100,4100,4000
4000  ERLST=RMSERR
      WRITE(LERR,3400) LOOP,RMSERR
      WRITE(LERR,4050) MXLOOP
4050  FORMAT(' MAXIMUM NUMBER ,',I3,', OF LOOPS CARRIED OUT.')
4100  CONTINUE
```



```
      WRITE(LERR,3400) LOOP,RMSERR
      RETURN
      END
```

Dec 16 09:44 1988 xpfil.f Page 1

```
      SUBROUTINE XPFIL(XF,YF,HRAN,NDATA,INCRAN,NAME,XG,YG,ER)
C
C             XPFIL IMPROVES THE AGREEMENT BETWEEN CALCULATED RANGES
C             AND OBSERVED RANGES BY ADJUSTING THE POSITION OF THE
C             POINT XG,YG SO THAT THE NDATA RANGES(HRAN) TO THE FIXED DATA
C             POINTS(TRANSPONDERS) XF,YF,HAVE A MINIMUM ERROR,ER.
C             INCRAN IS THE INCREMENT BETWEEN SUCCESSIVE
C             RANGES IN THE HRAN ARRAY.
C
      include 'xpcom.h'
C
      CHARACTER*6 NAME
C             NAME IS THE NAME OF THE TRANSPONDER OR TIME OF FIX.
      IMPLICIT REAL*4 (A-H, O-Z)
      DIMENSION XF(NDATA),YF(NDATA),HRAN(100)
C             XF,X-CO-ORDINATE OF FIXED POINT.
C             YF,Y-CO-ORDIMATE OF FIXED POINT.
C             HRAN,RANGE FROM POSITION TO FIXED POINT.
C
      DATA MLOOP/30/
C             MLOOP,MAXIMUM NUMBER OF LOOPS.
      DATA ERMIN/0.150/
C             ERMIN MAXIMUM SIZEOF ERROR ACCEPTABLE TO STOP LOOPING
      DATA REDUC/0.00015/
C             REDUC,MAXIMUM SIZE OF ERROR REDUCTION IN EACH LOOP TO
C             JUSTIFY STOPPING LOOPING.
      DATA GAIN,ZERO/1.50,0.010/
C             GAIN CONTROLS THE STABILITY OF THE CONVERGENCE PROCESS.
C             ZERO CONTROLS THE BEHAVIOUR WHEN CLOSE TO A TRANSPONDER.
C
C
      ERPR=10000.0
C             ERPR WILL CONTAIN ERROR SQUARED OF PREVIOUS LOOP.
      DO 4000 LOOP=1,MLOOP
C
      DX=0.0
      DY=0.0
C             DX AND DY ARE POSITION CORRECTION VECTOR COMPONENTS.
      FN=0.0
C             FN IS THE NUMBER OF VALID RANGES.
      ERR=0.0
C             ERR WILL CONTAIN CUMULATIVE SUM OF ERROR SQUARED.
C
C             CHECK TO SEE IF RANGE IS PRESENT.
      DO 1500 NDAT=NDATA,1,-1
C    ...SKIP IF  NO MEASURED RANGE IS AVAILIBLE.
      INDEX=(NDAT-1)*INCRAN+1
      HH = HRAN(INDEX)
CD    PRINT *, 'HH = ',HH
      IF(HH) 1500,1500,1100
      IF(XF(NDAT).EQ. 0.0 .AND. YF(NDAT).EQ. 0.0) GO TO 1500
1100  XDIFF=XG-XF(NDAT)
      YDIFF=YG-YF(NDAT)
      RNGEC=SQRT(XDIFF*XDIFF+YDIFF*YDIFF)
CD    PRINT *, 'HH,XDIF,YDIF,RNGEC=',HH,XDIFF,YDIFF,RNGEC
      IF(RNGEC-ZERO)1500,1500,1200
```


Dec 16 09:44 1988 xpfil.f Page 2

```
C
1200  RATIO=(HH-RNGEC)/RNGEC
      DX=RATIO*XDIFF + DX
      DY=RATIO*YDIFF + DY
      FN=FN+1.0
C
      ERR=ERR+(RNGEC-HH)**2
C     PRINT *,'RATIO,DX,DY,FN,ERR=',RATIO,DX,DY,FN,ERR
1500  CONTINUE
2000  IF(FN.LT..5) GO TO 8000
      ERR=SQRT(ERR/FN)
      STEP = 1.5
C      IF((ERPR-ERR)/ERPR .LT. 1.0) STEP=STEP*(ERPR-ERR)/ERPR
      XG=XG+DX*STEP/FN
      YG=YG+DY*STEP/FN
CD    WRITE(LFINL,2100)XG,YG,ERR,DX,DY
CD2100 FORMAT(' XG= ',F8.0,' YG= ',F8.0,' ERR=',
CD    1F8.1,' DX= ',F8.1,' DY=',F8.1)
      IF(ERR-ERMIN)8000,8000,3000
C             TEST TO SEE IF MAXIMUM TOLERABLE ERROR REDUCTION HAS OCCURRED.
3000  IF((ERPR-ERR)/ERPR-REDUC)8000,8000,3900
3900  ERPR=ERR
4000  CONTINUE
C             STOPPED LOOPING BECAUSE NUMBER OF ITERATIONS EXCEEDED.
      IF (LERR) 8000,8000,4100
4100  WRITE (LERR,4200)NAME,MLOOP
4200  FORMAT (1X,A,'FIX REQUIRED MORE THAN ',I3,' LOOPS')
8000  CONTINUE
C     WRITE (LINTER,3950) LOOP,ERPR, ERR
C3950 FORMAT (1X,I6,2X,F6.1,2X F6.1)
      ER=ERR
CD    WRITE (LFINL,8100) XG,YG,ER
CD8100 FORMAT(' RETURN FROM XFIL WITH XG=',F8.0,' YG=',F8.0,
CD    1' ER=',E12.2)
      RETURN
      END
```

```
      SUBROUTINE XPREAD (NAMFIL,OLDERR)
C     READS IN TRANSPONDER DATA FROM AN ARBITRARY FILE FOR LOOPING.
C
C
C             NAMFIL  ...Buffer containing name of data file.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      DIMENSION       RANS(20)
C                                             !Temporary array for ranges.
      CHARACTER*(*)   NAMFIL
C                                             !Will contain name of file being read.
C
      include 'xpcom.h'
C
      DATA LUDAT/16/
C
C      VM - ABS(DEEPV-1500.)
C      VF - ABS(DEEPV-1500.)
      VNOM - 1500.
C     IF(VM .LT. VF) VNOM - 1500.
      IF(NPOS .EQ. 0) OPEN (UNIT=LUDAT,FILE=NAMFIL)
C                                             !open file
C
      IF(NOMORE .GT. 0) GO TO 400
  100     CONTINUE
C                                             !Beginning of major loop.
          IF(NPOS .EQ. NPOMX)
     *        GO TO 300
          NPOS - NPOS + 1
          PSNS(NPOS,3)=0.
  105     CONTINUE
          DO 106 I=1,NTRS
  106         RANS(I) - 0.0
          READ (LUDAT,101,END=200) ITIME,
     *        (PSNS(NPOS,I),I=1,2),ERROR,DEPS,Q,
     *        (RANS(NTR),NTR=1,NTRS)
C
C          call xxcor(DEPS,0.0,VNOM,0.0,0,DEPS,DEPS)
          IF (DUMMY(2) .eq. 1.0) then
            do 107 i=1,NTRS
107           call xxcor(RANS(i),DEPS,VNOM,0.0,0,RANS(i),TRPN(i,3))
          ENDIF
C         WRITE (6,101) ITIME,
C    *        (PSNS(NPOS,I),I=1,2),ERROR,DEPS,Q,
C    *        (RANS(NTR),NTR=1,NTRS)
C
  101     FORMAT(11X,I5,2F7.0,F4.0,2F5.0,16F6.0)
C         VFCTR - DEEPV/VNOM
C         IF(DEPS .LT. 100.) VFCTR - SURFV/VNOM
          TIM(NPOS) - ITIME
C         PRINT *, ITIME,TIM(NPOS),(PSNS(NPOS,I),I=1,2),
C    *        ERROR,DEPS,Q,(RANS(NTR),NTR=1,NTRS)
C
          NRNGS - 0
          DO 110 NTR=1,NTRS
```



```
C                                             !save depths
              DEPTH(NPOS) - DEPS
C                                             !save rans in srans
              SRANS(NTR,NPOS) - RANS(NTR)
C                                             !for each transponder
              CRANS(NTR,NPOS) - 0.0
C                                             !clear horiz.  range
              IF(IMOVE(NTR) .EQ. 0) GO TO 110
C                                             !ignored range
              S - RANS(NTR)
C                                             !get slant range
              IF(S .EQ. 0.0) GO TO 110
C                                             !disregard if its 0
                  D - TRPN(NTR,3) - DEPS
C                                             !calc depth difference
                  IF(ABS(D) .GT. S)
C                                             !is it .GT. slant rng?
     *              WRITE(LERR,102) TIM(NPOS),
     *                  IXPCD(NTR),S,TRPN(NTR,3)
C                                             !yes, nasty message
                  IF(ABS(D) .GE. S) GO TO 110
C                                             ! and pass it by
                  C   - SQRT((S+D) * (S-D))
C                                             !get horiz range
                  IF(C .LT. 0.0 .OR. C .GT. 15000.)
     *                GO TO 105
C                                             !ignore a-physical rng
                  CRANS(NTR,NPOS) - C
C                                             !apply S.V. correction
                  NRNGS - NRNGS + 1
C                                             !inc'mt # of non-zero
  110         CONTINUE
C                                             ! ranges
          IF(NRNGS .LT. MINXPN) GO TO 105
C
CD        PRINT 111,TIM(NPOS),(CRANS(NTR,NPOS),NTR=1,NTRS)
          OLDERR=10000.
          GO TO 100
C                                             ! keep reading until buffer is
C                                                     full or file is empty.
  200     CONTINUE
          NOMORE - 1
C                                             !Flag that file is exhausted
          NPOS - NPOS - 1
C                                             !Didn't get one this time
C
  300     CONTINUE
C
  400 CONTINUE
      RETURN
  102 FORMAT(//,'**ERROR!**',1X,F8.4,' RANGE TO ',A2,' TRANSPONDER (',F8.2,
     *          ') LESS THAN TRANSPONDER DEPTH (',F8.2,' )',//)
CD111 FORMAT(16(1X,F14.2))
      END
```

```
Dec 17 12:56 1988  xpload.f Page 1

C       PROGRAM XPLOAD
C
        CHARACTER*48      COMENT(15)
        CHARACTER*80      LINE
        CHARACTER*1       ANSWER
        INTEGER           GTNUM
        LOGICAL           NEW, GETNAM, L1
        CHARACTER*80      IFILE
        CHARACTER*80      ITITLE
        CHARACTER*1       IYES
        CHARACTER*4       LABEL(15)
        REAL*4            TRX(15),TRY(15),TRZ(15)
        DIMENSION         IBUF(15), BUF(15)
C
        include 'xpcom.h'
C
        DATA              NCHAR/4/, IYES/'Y'/, TRMAX/15./
        DATA              ISCRN /6/, IKYBRD /5/
        DATA              ITITLE /' '/
        DATA              COMENT /15*' '/, LABEL /15*' '/
C
C
C
        LDFLG=0
C               FORCE XPCMLD TO READ FROM DISK INTO COMMON.
        CALL XPCMLD
CD      PRINT *, ' LFINL = ', LFINL
CD      PRINT *, ' LINTER = ', LINTER
CD      PRINT *, ' LERR = ', LERR
        PRINT *,' LSPAR =',LSPAR

        L1 = GETNAM('XPLOAD.LST',LSPAR,'N','Listing file : ',IFILE)
        OPEN(UNIT=LSPAR,FILE=IFILE,STATUS='UNKNOWN')
   10   CONTINUE
        IF(.NOT.GETNAM('TRANSPONDER.TRS',21,'OLD','Transponder file : ',
     *          IFILE)) GO TO 60
   21   FORMAT(A)
        OPEN (UNIT=19,STATUS='OLD',ERR=60,FILE=IFILE,
     1          FORM='FORMATTED')
C       ...EXISTING FILE NOW OPEN. INITIALIZE TRANSPONDER ARRAY
        NEW = .FALSE.
C
        READ(19,21,ERR=30) ITITLE
   30   CONTINUE
        NTRS = 0
        DO 40 I = 1,15
                READ(19,36,ERR=35,END=50) LABEL(I),
     1                    TRX(I),TRY(I),TRZ(I),COMENT(I)
                NTRS = NTRS+1
C                         DUMMY EXECUTABLE STMT FOR ERR
   35           IDUMY=I
   36           FORMAT(A,4X,2(F7.0,1X),F6.0,A)
   40           CONTINUE
   50   CLOSE (UNIT=19)
        GO TO 210
C
```

```
Dec 17 12:56 1988  xpload.f Page 2

   60   CONTINUE
        IF(ANSWER('Open new file ?') .NE. 'Y') GO TO 10
        NEW = .TRUE.
C       A NEW FILE IS TO BE CREATED...
  100   CONTINUE
        WRITE (*,'(''Descriptive title :'',$)')
        READ (*,'(a)') ITITLE
        NCH = LNBLNK(ITITLE)
        I =  GTNUM('Number of Transponders : ',A,1)
        IF(A.GT.TRMAX) A = TRMAX
        IF(A.LT.1.) GO TO 100
        NXPNDR = A
        NTRS = NXPNDR
        DO 110 I=1,NXPNDR
                IBUF(I) = I
  110           CONTINUE
C        LOOP READING DATA FOR XPNDRS WHOSE INDICES ARE IN IBUF
  120   CONTINUE
        DO 200 I = 1,NXPNDR
                ITR = IBUF(I)
  130           CONTINUE
                WRITE(ISCRN,131) ITR
  131           FORMAT(1X,'Input transponder ',1I2,'...'/)
                WRITE (*,'(''Label,X,Y,Depth,Comment : '',$)')
                READ (*,'(A)') LINE
                ICOMMA = INDEX(LINE,',')
C                          NO LABEL
                IF(ICOMMA .LE.1) GO TO 130
                LABEL(ITR) = LINE(1:ICOMMA-1)
C
                LINE = LINE(ICOMMA+1:)
                ICOMMA = INDEX(LINE,',')
                IF(ICOMMA .LE.1) GO TO 130
                TRX(ITR) = atof(LINE)
C
                LINE = LINE(ICOMMA+1:)
                ICOMMA = INDEX(LINE,',')
                IF(ICOMMA .LE.1) GO TO 130
                TRY(ITR) = atof(LINE)
C
                LINE = LINE(ICOMMA+1:)
                TRZ(ITR) = atof(LINE)
                ICOMMA = INDEX(LINE,',')
                IF(ICOMMA .LE.1) GO TO 200
C
                LINE = LINE(ICOMMA+1:)
                COMENT(ITR) = LINE
C
  200           CONTINUE
  210   CONTINUE
        WRITE(ISCRN,211) ITITLE, (I,LABEL(I),
     1          TRX(I),TRY(I),TRZ(I),COMENT(I),I=1,NTRS)
  211   FORMAT(/1X,A,/'  #  LABEL',6X,'X',6X,'Y',4X,'DEPTH',
     1  4X,'COMMENTS',//,100(1X,I2,2X,A,1x,3F7.0,1X,A,/))
C
        IF(ANSWER('Positions ok ?') .EQ. 'Y') GO TO 290
```

```
      IF(GTNUM('Enter numbers of positions to be revised : ',BUF,15)
     *        .EQ. 0) GO TO 290
      NXPNDR = 0
      DO 250 I=1,15
              IF(BUF(I).LE.0.0.OR.BUF(I).GT.TRMAX) GO TO 250
                      NXPNDR = NXPNDR + 1
                      IBUF(NXPNDR) = BUF(I)
                      NTRS = MAX0(IBUF(I),NTRS)
 250          CONTINUE
      IF(NXPNDR.GT.0) GO TO 120
 290  CONTINUE
      IF(.NOT.NEW) GOTO 292
      OPEN(UNIT=19,STATUS='NEW',FILE=IFILE,FORM='FORMATTED')
      GOTO 294
 292  OPEN(UNIT=19,STATUS='OLD',FILE=IFILE,FORM='FORMATTED')
 294  WRITE(19,296) ITITLE
 296  FORMAT(A)
      DO 295 I=1,NTRS
      WRITE(19,291) LABEL(I),TRX(I),TRY(I),TRZ(I),
     1                COMENT(I)
 291          FORMAT(X,A,4X,2(F7.0,1X),F6.0,A)
 295  CONTINUE
      CLOSE(UNIT=19)
 300  CONTINUE
      WRITE(LSPAR,301) IFILE
 301  FORMAT(' Transponder data file... ',A)
      WRITE(LSPAR,211) ITITLE,(I,LABEL(I),
     1        TRX(I),TRY(I),TRZ(I),COMENT(I),I=1,NTRS)
      DO 310 I = 1,NTRS
 310  IBUF(I) = I
      CALL BASLIN(LSPAR,NTRS,IBUF,LABEL,TRX,TRY)
      REWIND LSPAR
CD    PRINT *,'NTRS=',NTRS
      DO 410 I=1,NTRS
          TRPN(I,1) = TRX(I)
          TRPN(I,2) = TRY(I)
          TRPN(I,3) = TRZ(I)
          TRPN(I,4) = 0.0
          IXPCD(I) = LABEL(I)
CD        PRINT *,(TRPN(I,J),J=1,4)
 410      CONTINUE
 400  PRINT 411,SURFV,DEEPV
 411  FORMAT(' SURFV =',F8.1,' DEEPV = ',F8.1,$)
      IF(ANSWER(' ...Velocities OK ?') .eq. 'Y') go to 500
          I = GTNUM('Enter SURFV,DEEPV : ',SURFV,2)
          GO TO 400
 500  CONTINUE
      LDFLG = 1
      CALL XPCMLD
      CALL EXIT(0)
      END
```

```
Oct  5 15:27 1988  xprint.f Page 1

      SUBROUTINE XPRINT
C     XPRINT PRINTS OUT TIME, X,Y, AND ERROR FOR FIXES
C     INVOLVED IN THE LOOPING OPERATION.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      include 'xpcom.h'
C
      WRITE (LERR,4)
4     FORMAT('     TRX      TRY      TRZ        ID')
      DO 5 NTR=1,NTRS
      WRITE(LERR,6) (TRPN(NTR,I),I=1,3),IXPCD(NTR)
6     FORMAT(3F8.1,6X,A2)
5     CONTINUE
      WRITE(LERR,10)
10    FORMAT(///7X,'TIME',7X,'X',7X,'Y',6X,'ERROR'//)
      DO 20 I=1,NPOS
      T=TIM(I)
C     WRITE(LERR,15) I,T,(PSNS(I,J),J=1,3),(CRANS(NTR,I),NTR=1,NTRS)
15    FORMAT(1X,I3,1X,F6.1,2(1X,F8.1),1X,F7.3,15(1X,F8.2))
20    CONTINUE
      RETURN
      END
```

```
Feb 12 10:46 1985  basln2.f Page 1

      SUBROUTINE BASLN2(LU,NTRS,TRX,TRY,ALEN,NLEN,NAME)
C
C     VARIABLE USAGE
C
C           LU      ...Logical unit on which baselines are to be printed.
C           NTRS    ...Number of elements in TRX and TRY.
C           TRX     ...Buffer of position X-coordinates
C           TRY     ...Buffer of position Y-coordinates
C           ALEN    ...Buffer of permutations of distances between
C                      TRX(i),TRY(i) and TRX(j),TRY(j)
C                      (for i=2,NTRS,(j=1,i-1))
C           NLEN    ...Number of permuted lengths = (NTRS * (NTRS-1))/2
C           NAME    ...Two character transponder identifier
C
C
      IMPLICIT        REAL*4 (A-H, O-Z)
      DIMENSION       DIST(16), TRX(16), TRY(16)
      DIMENSION       ALEN(1), NAME(1)
C
CD    TYPE *, ' LU = ',LU
CD    TYPE *, 'NTRS = ',NTRS
CD    TYPE *, ' TRX(),TRY()=',(TRX(I),TRY(I),I=1,NTRS)
C
      IF(NTRS.LE.1) GO TO 400
C                                  !NOTHING TO DO... SO DON'T TRY
C
C   ...PRINT A LOWER TRIANGULAR MATRIX
C
      NLEN = 0
      WRITE(LU,101) (NAME(I),I=1,NTRS-1)
  101 FORMAT(3X,10(5X,A2,5X))
      DO 300 IROW = 2,NTRS
            XROW = TRX(IROW)
            YROW = TRY(IROW)
            NCOL = 0
C
            DO 200 ICOL = 1,IROW-1
                  NCOL = NCOL + 1
                  XCOL = TRX(ICOL)
                  YCOL = TRY(ICOL)
                  DX = XROW - XCOL
                  DY = YROW - YCOL
                  DIST(ICOL) = SQRT(DX*DX + DY*DY)
                  IF(XROW.EQ.0.0.AND.YROW.EQ.0.0
     *            .OR. XCOL.EQ.0.0.AND.YCOL.EQ.0.0)
     *                  DIST(ICOL) = 0.0
C                                        !IGNORE NONEXISTANT PSNS
                  NLEN = NLEN + 1
CD                TYPE *,'N,DIST=',NLEN,DIST(ICOL)
                  ALEN(NLEN) = DIST(ICOL)
  200             CONTINUE
            WRITE(LU,201) NAME(IROW),(DIST(I),I=1,NCOL)
  201       FORMAT(1X,A2,16(F12.5))
  300       CONTINUE
            WRITE(LU,301)
  301       FORMAT(/)
```

```
Feb 12 10:46 1985  basln2.f Page 2

  400 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE XPRIN2
C     XPRINT PRINTS OUT TIME, X,Y, AND ERROR FOR FIXES
C     INVOLVED IN THE LOOPING OPERATION.
C
      IMPLICIT REAL*4 (A-H, O-Z)
      CHARACTER*80    NEWOUT
C
      INTEGER K
      LOGICAL         GETNAM
C
      include 'xpcom.h'
C
      IF(.NOT.GETNAM('test.new',LNEW,'OLD','New position output: ',NEWOUT))
     * GOTO 60
      OPEN (UNIT=LNEW,STATUS='OLD',ERR=120,
     * FILE=NEWOUT,FORM='FORMATTED')
      GOTO 100
 60   OPEN (UNIT=LNEW,STATUS='NEW',ERR=120,
     * FILE=NEWOUT,FORM='FORMATTED')
 100  CONTINUE
C
      WRITE(LSCRN,10)
 10   FORMAT(' XPRINT ')
C
          DO 106 I=1,NPOS
          K = TIM(I)
          WRITE (LNEW,101) K,
     *        (PSNS(I,J),J=1,2),PSNS(I,3),DEPTH(I),Q,
     *        (SRANS(NTR,I),NTR=1,NTRS)
 106      CONTINUE
C
 101      FORMAT(11X,I5,2F7.0,F5.1,F4.0,F5.0,16F6.0)
      CLOSE(LNEW)
      RETURN
 120  PRINT *,'STOP: NO OUTPUT FILE'
      CALL EXIT(10)
      END
```

```
Dec 14 15:00 1987  xpurge.f Page 1

      SUBROUTINE XPURGE
C     XPURGE PICKS UP THE RMS ERROR FOR THE FIXES FROM COMMON
C     AND THEN PROCEEDS TO ELIMINATE ALL FIXES WHICH HAVE AN
C     ERROR GREATER THAN TWICE THE RMS ERROR.
C
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      include 'xpcom.h'
C
      LOGICAL GIVEN
C
      TOOBAD = DUMMY(1)
      GIVEN = .FALSE.
      NEW=0
      DO 2000 NPO=1,NPOS
      IF(PSNS(NPO,3) .GT .TOOBAD * RMSERR) GO TO 1500
      NEW=NEW+1
      IF(NEW.EQ.NPO) GO TO 2000
      DO 1200 NTR=1,NTRS
1200     CRANS(NTR,NEW)=CRANS(NTR,NPO)
      DO 1250 NTR=1,NTRS
1250     SRANS(NTR,NEW)=SRANS(NTR,NPO)
      DO 1300 IPOS=1,3
1300  PSNS(NEW,IPOS)=PSNS(NPO,IPOS)
      TIM(NEW)=TIM(NPO)
      DEPTH(NEW)=DEPTH(NPO)
      GO TO 2000
C
C
1500  CONTINUE
      IF(.NOT. GIVEN) WRITE(LERR,1501)
1501  FORMAT(//,'*** POSITION(S) REJECTED...***')
      GIVEN = .TRUE.
      T = TIM(NPO)
      WRITE(LERR,1502)T,(PSNS(NPO,J),J=1,3),
     *                   (CRANS(NTR,NPO),NTR=1,NTRS)
1502  FORMAT(1X,F6.1,2(1X,F8.1),1X,F7.3,15(1X,F8.2))
2000  CONTINUE
      NPOS=NEW
      RETURN
      END
```

```
      SUBROUTINE XPINPT (NAMFIL,IFILE,EFILE,ITIMES)
C
C             NAMFIL  ...Byte buffer containing data file name
C             IFILE   ...Byte buffer containing output XPNDR file.
C             ITIMES  ...Number of times that different statistical
C                        "realizations" of same data are to be found.
C
      IMPLICIT REAL*4 (A-H, O-Z)
C
      CHARACTER*(*)   NAMFIL
C                     !Will contain name of file being read.
      CHARACTER*(*)   IFILE
      CHARACTER*(*)   EFILE
      LOGICAL         GETNAM,LTEMP
      CHARACTER*80    STRING
      CHARACTER*1     IBLNK
C
C
      include 'xpcom.h'
C
C
      DATA IBLNK/' '/
      DATA LUDAT/17/
C
      IF(.NOT.GETNAM(' ',17,'OLD','Navigation data file : ',NAMFIL))
     *        GOTO 6000
      IF (NAMFIL.EQ.IBLNK) GO TO 6000
      OPEN (UNIT=LUDAT,FILE=NAMFIL)
      LTEMP = GETNAM('LOOPED.TRS',31,'NEW',
     *             'Looped output transponder file : ',IFILE)
C     PRINT *,'Looped output error file :  '
C     READ (*,'(a)') EFILE
C     OPEN (UNIT=LERR,FILE=EFILE,STATUS='NEW')
C
      MINXPN = 3
      CALL GTNUM('Minimum no. of ranges acceptable ? ',A,1)
      IF( A .GT. 1) MINXPN = A
C
C
      PRINT 1405, (IXPCD(NTR),NTR=1,NTRS),('  ',I=1,8-NTRS)
1405  FORMAT(8(X,A2),' transponders are currently in the system.',/
     1 ,' If you wish to adjust all positions,just press "return".'/
     2 ,' If you wish to select particular transponders for',/
     3 ,' looping, type in their names one at a time followed',/
     4 ,' by "return". To terminate the list of names,',/
     5 ,' press "return" without entering a name.'//)
C
      DO 1410 NTR=1,NTRS
1410     IMOVE(NTR)=1
C                      !IMOVE IS ONE WHEN POSITION IS TO BE CHANGED.
C
      WRITE(*,'(''Transponder name :'',$)')
      READ (*,'(a)') STRING
      IF(LNBLNK(STRING) .LE. 1) GO TO  1490
      DO 1440 NTR=1,NTRS
1440     IMOVE(NTR)=0
      GO TO 1450
```



```
C
1430  CONTINUE
      WRITE(*,'(''Transponder name :'',$)')
      READ (*,'(a)') STRING
      IF(LNBLNK(STRING) .LE. 1) GO TO  1490
1450  CONTINUE
1431  FORMAT(A2)
CD    PRINT 1431,' ',string
      DO 1460 NTR=1,NTRS
          IF(STRING.EQ.IXPCD(NTR)) GO TO 1470
1460      CONTINUE
C
      PRINT 1465 ,STRING
1465  FORMAT(2X,A2,' Not recognized as valid transponder name')
      GO TO 1430
C
1470  CONTINUE
CD    PRINT *,'XPNDR IDENTIFIED...'
      IMOVE(NTR)=1
CD    PRINT *,'IMOVE=',(IMOVE(I),I=1,NTRS)
      GO TO 1430
C
1490  CONTINUE
C
CD    PRINT *,'IMOVE=',(IMOVE(I),I=1,NTRS)
CD    PRINT *
      RETURN
 6000 PRINT *,'STOP: no data file specified'
      CALL EXIT(10)
      END
```

```
      subroutine tmdate(datstr,timstr)
c
      character*(*)   datstr,timstr
      character*24    chtime
      character*24    ctime
c
      integer*4       iuxtim,time
c
      iuxtim = time()
      chtime = ctime(iuxtim)
      datstr = chtime(9:10) // '-' // chtime(5:7)
     *                // '-' // chtime(23:24)
      timstr = chtime(12:19)
c
      return
      end
```

```
May 18 16:00 1987  atof.c Page 1


/*
 *      ATOF    fortran callable version of 'C' routine atof
 */
int atof_(string)
char *string;
{
        double atof();   float temp;
        temp= (float) atof(string);
        return *((int *)&temp);
}
/*
 *      ATOI    fortran callable version of 'C' routine atoi
 */
```

```
Jun  3 15:27 1987  xpcom.h Page 1


C
      EQUIVALENCE     (SPARE(1),RMSERR)
      EQUIVALENCE     (ISPAR(1),NOMORE),(ISPAR(2),MINXPN)
C
      COMMON          /MOVE/IMOVE(15)
C
      CHARACTER*4 IXPCD(15)
C
      COMMON/XPCOM/SPARE(15),TRPN(15,4),
     1         SCALE,XL,XR,YB,YT,XO,YO,
     2         SURFV,DEEPV,DUMMY(26),
     3         ISPAR(15),LFINL,NTRS,LINTER,LERR,LSPAR,LDFLG,IXPCD,
     4         LSCRN,LNEW
C
      COMMON/XPDATA/NPOMX,NPOS,CRANS(6,1500),PSNS(1500,3),TIM(1500),
     1         SRANS(6,1500),DEPTH(1500)
C
```

Dec 20 10:30 1988 Makefile Page 1

```
OBJECTS = harrynav.o navloc.o fillnav.o navtst2.o reads.o prints.o
CFLAGS =  -O
LIBES = -lsiofpa
FILENAME = savwin
SOURCES = harrynav.c navloc.c fillnav.c navtst2.c reads.c prints.c
$(FILENAME): $(OBJECTS)
	cc  $(CFLAGS) $(OBJECTS) -o $(FILENAME) $(LIBES)
	size $(FILENAME)
clean:
	-rm *.o
print:
	@echo Listing updated source files:
	@echo $?
	print $?
	touch $@
harrynav.o: nav.h
fillnav.o: nav.h
navtst2.o: nav.h
navloc.o: nav.h
reads.o: nav.h
prints.o: nav.h
```

NAME

harrynav - convert a MPL horizontal array data file to a NAV SIO file

SYNOPSIS

harrynav [-abptv] nskproll nroll t/lstroll procfile flipfile outnavfile

DESCRIPTION

Harrynav converts a MPL horizontal array data file into a navigation SIO data file. lstroll is the first rollover to use (the first rollover is number 1 (unless the -t flag is chosen). Nskproll is the number of rollovers to skip prior to extracting more. Nroll is the total number of rollovers to extract; if < 1 all rollovers are extracted until you respond with a 'q' when prompted to mount a tape. Procfile is a sio data file consisting of processor numbers (bottom one is the 1st) used to check the data file. Note that each rollover returns 4 transponders of information each of which contains procfile processors which contain 3 navigation words each (size, time and range) (* 2 channels if -b case). Flipfile is a sio file of Flip positions.

OPTIONS

-a indicates the user is to be prompted for an ASCII string which will then be inserted in the comment block in the output file header record.

-b indicates format 6B input records and corresponding 28 channel navfile output (see appendix).

-p indicates to print all 1's present in the nav data.

-t indicates to use nstart as a time string and to use the first record that exceed this time. This string is in the format "hr:min:sec:msec" and buffers whose goes clock (in the buffer header) are less than this are skipped.

-v indicates verbose option; to print the results in gory detail.

LANGUAGE: C

BUGS: Due to the desire to keep the program simple, all possible erroneous input conditions have not been guarded against.

DATA TAPE STRUCTURE: The MPL horizontal array data tapes consist of 16 header bytes + 64 * (4 + #processors* 16) byte records (12560 bytes in 1987 version) which have a structure as defined in the appendix below. Every record is subdivided into a 8 word buffer header (the first of which is placed in the first 8 words of the sio user variables) and 64 frames each of which contain a two header words and procfile processor blocks. Each processor block contains either 2 (format 6B) 12 bit or 1 (format 6A) 5 bit navigation samples. In the first case the high 10 bits are used; in the second pairs of 5-bit words are packed into 10-bit words which are then processed. The data were taken by an 12 bit A/D converter operating at a sampling rate of 500 Hz.

TAPE READING

The program reads directly from the tape drives. The user is prompted to load the tapes and put them on line, then specify the drive number.

```
#define MAXPROCS 12
#define NUMBUF 74
#define NWPBa 32
#define NWPBb 64
#define SAVSZ 2500 /* number of bits in 1 second plots */
#define WINOFF 10  /* Number of words to offset start by */
/* Length of window in bits */
static int LENGTH[48]= {250,250,250,250,250,250,250,250,250,250,250,250,
                        625,625,625,625,625,625,625,625,625,625,625,625,
                        625,625,625,625,625,625,625,625,625,625,625,625,
                        250,250,250,250,250,250,250,250,250,250,250,250};
float savsuml[48*2500];
union u_tag {
        unsigned short sbuf[ 8+32*(4+MAXPROCS*16)];
        unsigned char  cbuf[16+64*(4+MAXPROCS*16)];
    } u;
int cntbuf;
int istart[4][12];
```

```
Dec 20 10:29 1988  harrynav.c Page 1

#include <math.h>
#define FALSE     0
#define TRUE      -1
#include "nav.h"
main(argc,argv)
int argc; char *argv[];
/* C main program: read in parameters from usage line, check them */
    {
    float r[3],a[65536];   char *strs[4];  int NPROC,nc,nstart,ichans[14],i;
    int bo[5],errflg,istuff[64],rl,nskproll,nroll;  unsigned char procnos[14];
    static char *bi[] = {"-a","-b","-p","-t","-v"};   char dstr[71];
    static char *cstr={
"Usage: harrynav [-abptv] nskproll nroll t/lstroll procfile flipfile outnavfile"
    prompts(cstr,5,bi,bo,2,4,r,strs,dstr,&errflg,argc,argv);
    if (errflg)exit(1);
    nskproll= (int) r[0];  if (nskproll < 1) nskproll = 1; /* skip #rollovers */
  /*  Nstart is the first rollover to start at */
    nroll= (int) r[1];  if (nroll < 1) nroll= 5000;
    ncsetup(&NPROC,&nc,ichans,strs[1]); /* Read in proc. numbers */
    rdsio(a,NPROC,1,1,ichans,&rl,strs[1],istuff);
    for (i=0; i<NPROC; i++) procnos[i]=a[i];
    nstart=1;  if (!bo[3]) { nstart= atof(strs[0]); if (nstart < 1) nstart=1; }
    if (~errflg)
        getnav(nstart,nskproll,nroll,NPROC,procnos,strs[0],strs[2],strs[3],bo[1]
               bo[2],bo[3],bo[4]);
    }

#define RLO 1024
#include <sys/file.h>
#include <stdio.h>
getnav(nstart,nskproll,ntroll,NPROC,procnos,tparam,flipfn,navfn,bflg,pflg,tflg,v
int    nstart,nskproll,ntroll,NPROC,                            bflg,pflg,tflg,v
unsigned char procnos[];            char   tparam[],flipfn[],navfn[];
    {
    int i,j,k,fd=0,start,usi[64],nflip,nflipc,ichans[5],rl,navch,il,
        bufsiz,istuff[64],nroll=0,time,timel,pst[12];  unsigned short hr,hrt;
    float nout[RLO*20],a[424*5],plotwin[192];  FILE *fp,*fpi;
        char gmtstr[18],sysstr[80];  int bwin[4];
    ncsetup(&nflip,&nflipc,ichans,flipfn); /* Read in proc. numbers */
    rdsio(a,nflip,5,1,ichans,&rl,flipfn,istuff);
    navch=1;  if (bflg) navch=2;  cntbuf=0; bufsiz = 16 + 64 * (4 + NPROC*16);
    fp= fopen(navfn,"w");
    fd= whichtp(vflg,nout,nroll,NPROC,procnos,navch,fd,fp);
    skipread(nstart,0,&fd,bufsiz,&start,vflg,nout,nroll,NPROC,procnos,navch,fp);
    if (tflg)
      { hr= (u.sbuf[5]>>8)&0x1f;  hr += 7;  if (hr >= 24) hr -= 24;
        time= ((int)((u.sbuf[2]&0x0fff)<<16)) | u.sbuf[7];
        timel= gettime(tparam,&hrt);
        while  ((time < timel) && (hr <= hrt))
          { skipread(nstart,0,&fd,bufsiz,&start,vflg,nout,nroll,NPROC,procnos,
                    navch,fp);
            hr= (u.sbuf[5]>>8)&0x1f;  hr += 7;  if (hr >= 24) hr  -= 24;
            time= ((int)((u.sbuf[2]&0x0fff)<<16)) | u.sbuf[7]; }          }
/* Create nav output file header */  for (i=0; i<8; i++) usi[32+i]=u.sbuf[i];
    for (i=0; i<ntroll; i++)
        {
```

```
Dec 20 10:29 1988  harrynav.c Page 2

        if ((fpi= fopen("window.dat","r")) == NULL)
          error(" Error opening input file ","window.dat");
        for(j=0;j<4;j++)
           { for (k=0;k<12;k++) fscanf(fpi,"%d ",&istart[j][k]);
             fscanf(fpi,"\n"); }
        fclose(fpi);
        dtfpos(a,fp,nflip,-1,gmtstr);
        for (j=0; j<192; j++) plotwin[j]=0;
        for (j=0; j<4; j++) /* For 4 transponders */
            {   /* compute start, call program which calls navloc */
            if (pflg) printf(" Rollover# %d,  Transponder# %d\n",nroll+1,j+1);
            il=(nroll*4 +j)* NPROC*3*navch;
            if (j > 0) skipread(0,10000*j,&fd,bufsiz,&start,vflg,nout,nroll,
                  NPROC, procnos,navch,fp);
            if (!bflg) fillnav(start,&nout[il],j,&fd,NPROC,procnos,nroll,pflg,
                  vflg,nout,navch,fp,pst,&bwin[j]);
            else  fillnavb(start,&nout[il],j,&fd,NPROC,procnos,nroll,pflg,
                  vflg,nout,navch,fp);
            for (k=0; k<12; k++)
              { plotwin[(j*12+k)*2]=pst[k]*.4;
                plotwin[(j*12+k)*2+1]=plotwin[(j*12+k)*2]+ LENGTH[j*12+k]* .4; }
            }
        mksio(plotwin,96,2,384,"pltwin.sio"," ",istuff);
        mksio(savsuml,SAVSZ*12*4,1,SAVSZ,"navplt.sio"," ",istuff);
                sprintf(sysstr,"pltsav.scr %4d %s %6d %6d %6d %6d",i+1,gmtstr,
                                 bwin[0],bwin[1],bwin[2],bwin[3]);
                printf("%s\n",sysstr);
        if ((fpi= fopen("window.dat","w")) == NULL)
          error(" Error opening output file ","window.dat");
        for(j=0;j<4;j++)
         { for (k=0;k<12;k++) fprintf(fpi,"%d ",istart[j][k]);
                                 fprintf(fpi,"\n"); }
                dtfpos(a,fpi,nflip,0,gmtstr);
        fclose(fpi);
        /* system ("pltsav.scr i xx 1 2 3 4"); */
        for (k=0; k<NPROC*navch; k++) /* For NPROC*navch processors*/
            { il= nroll*4*NPROC*3*navch + k*3 + 1;
              fprintf(fp,"%10.3f %10.3f %10.3f %10.3f\n",nout[il],
                nout[il+NPROC*3*navch], nout[il+2*NPROC*3*navch],
                nout[il+3*NPROC*3*navch]); }
        fflush(fp); /* flush to make sure records are written */
        system (sysstr);
        nroll++;
        skipread(nskproll,0,&fd,bufsiz,&start,vflg,nout,nroll,NPROC,procnos,
                   navch,fp);  /* go to next rollover */
        }
/* NPROC processors, (size,time,range), 4 transponders, nroll rollovers */
   /* mksio(nout,NPROC*3*4*nroll*navch,1,rlo4,navfn,dstr,usi); */
   }
```

Dec 9 20:03 1988 reads.c Page 1

```c
#define FALSE      0
#define TRUE      -1
#include <ctype.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mtio.h>
#include <stdio.h>
#include "nav.h"
skipread(nrollskip,hwclk,fd,bufsiz,start,vflg,nout,nroll,NPROC,procnos,navch,fp)
int     nrollskip,hwclk,*fd,bufsiz,*start,vflg,    nroll,NPROC,        navch;
float nout[]; unsigned char procnos[]; FILE *fp;
    {
    int nrskip=0,rflg=FALSE; unsigned short newroll=65536-127,oseq;
    oseq=u.sbuf[1];
    while (nrollskip > nrskip) /* skip to nrollskip rollover */
        {
        if (read(*fd,(char *)u.cbuf,bufsiz) < bufsiz) /* suspect! */
          { /* assume EOT switch tapes, read */
          cntbuf++;  printf("Error - Partial buffer! - cntbuf= %d\n",cntbuf);
          if (read(*fd,(char *)u.cbuf,bufsiz) == bufsiz) /* OK! */
            { swab((char *)u.cbuf, (char *)u.cbuf,4);  cntbuf++;
              if (u.sbuf[1]== (oseq+2)) printf("Skipped Buffer# %d\n",cntbuf-1);
              else  { oseq= u.sbuf[1];  printf(
                                         " Error - two in a row try new tape\n"
          }
        else  swab((char *)u.cbuf, (char *)u.cbuf,4);
        if (u.sbuf[1] == oseq)
          { /* same seq.#s assume EOT */
          *fd=whichtp(vflg,nout,nroll,NPROC,procnos,navch,*fd,fp);
          if (read(*fd,(char *)u.cbuf,bufsiz) < bufsiz)
            { printf("Fatal tape read error cntbuf= %d\n",cntbuf);  exit(1);}
          swab((char *)u.cbuf, (char *)u.cbuf,4); cntbuf++;
          }
        if ((u.sbuf[0] > newroll) || (u.sbuf[0] < 2))  nrskip++;
        rflg=TRUE;  cntbuf++;  oseq= u.sbuf[1];
        }
    while (hwclk > (u.sbuf[0]+127))
        {
        if (read(*fd,(char *)u.cbuf,bufsiz) < bufsiz) /* suspect! */
          { /* assume EOT switch tapes, read */
          cntbuf++;  printf("Error - Partial buffer! - cntbuf= %d\n",cntbuf);
          if (read(*fd,(char *)u.cbuf,bufsiz) == bufsiz) /* OK! */
            { swab((char *)u.cbuf, (char *)u.cbuf,4);  cntbuf++;
              if (u.sbuf[1]== (oseq+2)) printf("Skipped Buffer# %d\n",cntbuf-1);
              else  { oseq= u.sbuf[1];  printf(
                                         " Error - two in a row try new tape\n"
          }
        else  swab((char *)u.cbuf, (char *)u.cbuf,4);
        if (u.sbuf[1] == oseq)
          { /* same seq.#s assume EOT */
          *fd=whichtp(vflg,nout,nroll,NPROC,procnos,navch,*fd,fp);
          if (read(*fd,(char *)u.cbuf,bufsiz) < bufsiz)
            { printf("Fatal tape read error cntbuf= %d\n",cntbuf);  exit(1);}
          swab((char *)u.cbuf, (char *)u.cbuf,4); cntbuf++;
          }
```

Dec 9 20:03 1988 reads.c Page 2

```c
        rflg=TRUE;  cntbuf++;  oseq= u.sbuf[1];
        }
    if (rflg) swab((char *)&u.cbuf[4],(char *)&u.cbuf[4],bufsiz-4);
    if (hwclk == 0) *start= (65536 - u.sbuf[0]);
    else            *start= (hwclk - u.sbuf[0]);
    }

whichtp(vflg,nout,nroll,NPROC,procnos,navch,fdo,fp)
int vflg,nroll,NPROC,navch,fdo;  float nout[];  unsigned char procnos[];
FILE *fp;
  {
  struct mtop mtop;  char *fin,c;  int fd;
  if (fdo > 0)
     { mtop.mt_op= MTOFFL;  mtop.mt_count=1;  ioctl(fdo,MTIOCTOP,&mtop);
       close(fdo);  }
  fd=0;
  while (fd < 1)
    {
    printf("\nLoad tape(online!) Then enter 0,1 or q: ");
    fflush(stdin);  scanf("%c",&c);
    while ((c != '0') && (c != '1') && (c != 'q'))
        { printf("\nEnter 0,1 or q: ");  fflush(stdin);  scanf("%c",&c); }
    if (vflg && (c == 'q'))
      { printnavp(nout,nroll,NPROC,procnos,navch); /* Print by processor */
        printnavt(nout,nroll,NPROC,procnos,navch); /* Print by Transponder*/ }
    if (c == 'q') { fclose(fp); exit(1); }
    if (c == '0') fin= "/dev/rmt0";  if (c == '1') fin= "/dev/rmt1";
        /* fin= "d946"; Temporary for use with disk file! */
    if ((c == '0') || (c == '1')) fd= open(fin,O_RDONLY);
    }
  return(fd);
  }

gettime(tparam,hrt)
char tparam[];  unsigned short *hrt;
{
int h,m,s,ms;  unsigned char m1,m2,s4,s5,ms1,ms2,ms3;  char s1,s2,s3;
sscanf(tparam,"%d%c%d%c%d%c%d",&h,&s1,&m,&s2,&s,&s3,&ms);
/* printf("%x %x %x %x \n",h,m,s,ms); */
if (h >= 24) h=24; *hrt= h;   /* Only 24 hours in our days! */
if (m > 100) m=99;  m2= (m/10);  m1 = (unsigned char) (m-m2*10);
if (s > 100) s=99;  s5= (s/10);  s4 = (unsigned char) (s-s5*10);
if (ms > 999) ms=999;  ms3= (ms/100);  ms2= ((ms-ms3*100)/10);
ms1= ms-ms3*100-ms2*10;
return ( (m2 << 24) | (m1 << 20) | (s5 << 16) | (s4 << 12)
       | (ms3 << 8) | (ms2 << 4) | (ms1) );
}
```

```
#include "nav.h"
#include <stdio.h>
#include <sys/file.h>
printnavt(nout,nroll,NPROC,procnos,navch)
unsigned char procnos[]; float nout[]; int nroll,NPROC,navch;
    {
    int i,j,k;
    for (i=0; i<4; i++)
      {
    /*  printf("INPUT FILE: %s\n",fin); */
      printf("TRANSPONDER %d\n",i+1);
      printf("NEWFILE --> %u",(u.sbuf[4] >> 8) & 0xf);
      printf("/%u/87",u.sbuf[4] & 0x1f);
      printf(" %u",(u.sbuf[5] >> 8) & 0x1f);
      printf(":%u",u.sbuf[5] & 0x3f); printf(":%u",u.sbuf[6] & 0x3f);
      printf(" %u\n",u.sbuf[1]);
      if (navch == 2)
        printf("  PROC  ROLL  TIME    RANGE    SIZ    TIME    RANGE    SIZ\n");
      else
        printf("  PROC  ROLL  TIME    RANGE    SIZ\n");
      for (j=0; j<NPROC; j++)
        for (k=0; k<nroll; k++)
          if (navch == 2)
            print2((int) procnos[j],k,&nout[k*NPROC*24 + j*6 + i*NPROC*6]);
          else
            print1((int) procnos[j],k,&nout[k*NPROC*12 + j*3 + i*NPROC*3]);
      }
    }

print1(il,jl,r)
int il,jl; float r[3];
    {
    printf(" %4d  %4d  %7.1f  %7.1f  %4d\n",il+1,jl+1,r[0],r[1],(int) r[2]);
    }

printnavp(nout,nroll,NPROC,procnos,navch)
unsigned char procnos[]; float nout[]; int nroll,NPROC,navch;
    {
    int i,j,k; /* ,point; */
    for (i=0; i<NPROC; i++)
      {
    /*  printf("INPUT FILE: %s\n",fin); */
      printf("PROCESSOR %d\n",procnos[i]);
      printf("NEWFILE --> %u",(u.sbuf[4] >> 8) & 0xf);
      printf("/%u/87",u.sbuf[4] & 0x1f);
      printf(" %u",(u.sbuf[5] >> 8) & 0x1f);
      printf(":%u",u.sbuf[5] & 0x3f); printf(":%u",u.sbuf[6] & 0x3f);
      printf(" %u\n",u.sbuf[1]);
      if (navch == 2)
        printf("  ROLL  XPDR  TIME    RANGE    SIZ    TIME    RANGE    SIZ\n");
      else
        printf("  ROLL  XPDR  TIME    RANGE    SIZ\n");
      for (j=0; j<nroll; j++)
        for (k=0; k<4; k++)
          if (navch == 2)
            print2(j,k,&nout[k*NPROC*6 + j*24*NPROC + i*6]);
```



```
          else
            print1(j,k,&nout[k*NPROC*3 + j*12*NPROC + i*3]);
      }
   /* for (i=0; i<nroll; i++)
    *   {
    *   printf("Roll * %d: ",i);
    *   for (j=0; j<4; j++)
    *     {
    *     printf("Transponder# %d: ",j+1);
    *     for (k=0; k<NPROC; k++)
    *       {
    *         printf("Processor# %d:",k+1);
    *         point= k*4*NPROC*3 + j*NPROC*3 + k*3;
    *         printf("%f %f %d \n",nout[point],nout[point+1],
    *             nout[point+2]);
    *       }
    *     }
    *   } */
    }

print2(il,jl,r)
int il,jl; float r[6];
    {
    printf(" %4d  %4d  %7.1f  %7.1f  %4d  %7.1f  %7.1f  %4d\n",il+1,jl+1,
        r[0],r[1],(int) r[2],r[3],r[4],(int) r[5]);
    }

dtfpos(a,fp,NT,pall,gm)
float a[]; FILE *fp; int NT,pall; char *gm;
  {
  short julian,hr,min,hrmin,hl,ml,h0,m0,dt,dtt;   float c,cl;  int i;
  if (((u.sbuf[4] >> 8) & 0xf) == 9) julian=244;
  if (((u.sbuf[4] >> 8) & 0xf) == 10) julian=274;
  julian += ((u.sbuf[4] & 0x1f) -1);  hr= (u.sbuf[5] >> 8) & 0x1f;  hr +=7;
  if (hr >= 24) { julian++;  hr -= 24; }
  sprintf(gm,"%d %2u:%u%u:%u%u.%u%u%u%u",julian,hr,((u.sbuf[2] >>  8) & 0xf),
    ((u.sbuf[2] >>  4) & 0xf),(u.sbuf[2] & 0xf),((u.sbuf[7] >> 12) & 0xf),
    ((u.sbuf[7] >>  8) & 0xf),((u.sbuf[7] >> 4) & 0xf),(u.sbuf[7] & 0xf));
  fprintf(fp,"%d",julian);  fprintf(fp," %2u",hr);
  fprintf(fp,":%u",(u.sbuf[2] >>  8) & 0xf);
  fprintf(fp, "%u",(u.sbuf[2] >>  4) & 0xf);  fprintf(fp,":%u",(u.sbuf[2]) & 0xf
  fprintf(fp, "%u",(u.sbuf[7] >> 12) & 0xf);
  fprintf(fp,".%u",(u.sbuf[7] >>  8) & 0xf);
  fprintf(fp, "%u",(u.sbuf[7] >> 4) & 0xf);
  fprintf(fp, "%u",(u.sbuf[7]      ) & 0xf);
  min= (u.sbuf[2] >> 4) &0xff; hrmin= hr*100+min;
  if (!pall) { fflush(fp); return; }
  for (i=0; i<NT; i++)
    { if (julian < a[i]) break;
        else if ((julian == a[i]) && (hrmin < a[i+NT])) break;  }
  hl=((int)(a[i+NT]/100))*100;  ml=a[i+NT]-hl;
  hl /= 100; h0=((int)(a[i-1+NT]/100))*100;  m0=a[i-1+NT]-h0;  h0 /= 100;
  dt=(a[i]-a[i-1])*24*60 + (hl-h0)*60 + ml-m0;
  dtt= (a[i]-julian)*24*60 + (hl-hr)*60 + ml-min;
  cl= (float)(dtt)/dt;  c= 1-cl;
  fprintf(fp," %10.3f"  ,c*a[i+2*NT]+cl*a[i-1+2*NT]);
```

```
fprintf(fp," %10.3f"  ,c*a[i+3*NT]+c1*a[i-1+3*NT]);
fprintf(fp," %10.3f"  ,c*a[i+4*NT]+c1*a[i-1+4*NT]);
c=89.92;  fprintf(fp," %10.3f\n",c);  fflush(fp);  return;
}
```

```
#include <stdio.h>
#include "nav.h"
fillnav(start,navbuf,l,fd,NPROC,procnos,rolln,pflg,vflg,nout,navch,fp,pst,bw)
int     start,      l,*fd,NPROC,        rolln,pflg,vflg,     navch,pst[12],*bw;
float navbuf[MAXPROCS*3],nout[];   unsigned char procnos[MAXPROCS];
FILE *fp;
    {
    int i=0,j=0,k=0,joff,jl=0,ib;
    int kout,kin,jout,bufsize,nfrerr,tnfrerr,nbadproc,nbadfr;
    unsigned short SYNC= 0xeb90,stemp,sbuf[NUMBUF*NWPBa*MAXPROCS+NWPBa],
       oseql,s;   unsigned char fctr,ofctr,ctemp;
    bufsize= 16 + 64* (4 + NPROC*16);
    oseql = u.sbuf[l]+1;
    printf ("\ncntbuf: %d, start: %d", cntbuf,start);
    printf ("\nhwclock: %d,  SEQN: %d\n",u.sbuf[0],u.sbuf[l]);
    tnfrerr=0;
    for (i=0; i<NUMBUF; i++) /* Get NUMBUF buffers (>9 sec) for each receiver*/
      {
      nbadfr=nbadproc=0;
      ofctr= ((u.cbuf[19] &0x7f) -1)&0x7f;
      for (j=0; j<64; j++)     /* get 64 (pack into 32) samples */
        { /* Note: we fill array from beginning of buffer, then pass
            start= offset into the buffer to start in msecs. */
        joff=16+j*(4+NPROC*16);     /* to frame */  ofctr=(ofctr+1)&0x7f ;
        /* frame output pointer (output frames - 1/2 input frames)  */
        stemp= u.sbuf[joff/2];
        if (stemp != SYNC)
            { fprintf(stderr," Error - Incorrect value of Frame Sync = %x\n",
               stemp); fprintf(stderr," Frame# %d\n",j+1);
              nbadfr++; break;   }
        fctr = u.cbuf[joff+3] & 0x7f;  nfrerr= fctr-ofctr;
        if (nfrerr < -64) nfrerr += 128;
        if (nfrerr < 0)
             { printf(
             "Error - Frame counter difference negative (= %d %d)",fctr,ofctr);
             if (nfrerr != -1) exit(1);   }
        if (nfrerr > 0)
          {
          fprintf(stderr,"Counter incorrect! = %d, should = %d\n",fctr,ofctr);
          ofctr=fctr;
          for (jl=0; jl<nfrerr; jl++)
            for (k=0; k<NPROC; k++)
              { /* kout= output pointer for each processor */
               /* jout= frame output ptr - output frames = 1/2 input frames */
              kout = k*NUMBUF*NWPBa + i*NWPBa; jout=(j+jl+tnfrerr)/2;
              if (((j+jl)%2)==0)  sbuf[jout+kout] = 0;
            /* else sbuf[jout+kout] |= 0x0 << 3;}  unnecessary - already =0  */
              }
          tnfrerr += nfrerr;
          }
        for (k=0; k<NPROC; k++)    /* for NPROC Processors */
          {
          jout=(j+tnfrerr)/2; kout = k*NUMBUF*NWPBa + i*NWPBa;
  /* input pointer for each processor kin */
          kin=k*16+4+15; ctemp=u.cbuf[joff+kin] & 0xf8;
          if ((u.cbuf[joff+kin] & 0x07) != (procnos[k] & 0x07))
```



```
            {
            nbadproc++;
            if (k==0) ctemp = 0;
            else { if ((u.cbuf[joff+kin-16] & 0x07) == (procnos[k] & 0x07))
                     ctemp= u.cbuf[joff + kin - 16] & 0xf8;
                   else ctemp = 0; }
            }
          if (((j+tnfrerr)%2)==0)
              sbuf[jout+kout] =  (unsigned short)ctemp << 8;
          else  sbuf[jout+kout] |= (unsigned short)ctemp << 3;
          } /* End of k loop */
        } /* End of j loop */
      if(nbadfr > 0) fprintf(stderr,"# Bad frame sync's  = %d  Buffer# %d \n"
                         ,nbadfr,cntbuf);
      if(nbadproc > 0) fprintf(stderr,"# Bad processor ID's = %d  Buffer# %d \
                         ,nbadproc,cntbuf);
      cntbuf++;
      if (read(*fd,(char *)u.cbuf,bufsize) < bufsize)
        { *fd= whichtp(vflg,nout,rolln,NPROC,procnos,navch,*fd,fp);
          if (read(*fd,(char *)u.cbuf,bufsize) < bufsize)
           {printf("Fatal tape read error cntbuf= %d\n",cntbuf);  exit(1); } }
      swab((char *)u.cbuf,(char *)u.cbuf,bufsize); /* For Sun */
      if (u.sbuf[l] == (oseql-1))
          { printf(" Switch tapes;  cntbuf= %d\n",cntbuf);
            *fd= whichtp(vflg,nout,rolln,NPROC,procnos,navch,*fd,fp);
            if (read(*fd,(char *)u.cbuf,bufsize) < bufsize)
              {printf("Fatal tape read error cntbuf= %d\n",cntbuf);  exit(1);
            swab((char *)u.cbuf,(char *)u.cbuf,bufsize); /* For Sun */
            }
      if (u.sbuf[l] != oseql)
          {  fprintf(stderr,
             "Error - incorrect SEQN, should be %d, is %d, Buffer# %d \n",
              oseql,u.sbuf[l],cntbuf);
            for (j=0; j<NWPBa; j++)
              for (k=0; k<NPROC; k++)
                { kout = k*NUMBUF*NWPBa + (i+1)*NWPBa; sbuf[j+ kout]= 0;  }
          }
      oseql= u.sbuf[l]+1;
      } /* End of i loop */
    navloc(sbuf,navbuf,start,NPROC,l,NPROC,pst,bw);
    if (l == 3)  for (ib=0; ib<NPROC; ib++) navbuf[ib*3+1] += 89.92;
    printf ("\nRoll: %d,  Xponder: %d  Start: %d\n",rolln+1,l+1,start);
    for (ib=0; ib<NPROC; ib++)
        printf("Proc# %d;  %f  %f  %d\n",ib,navbuf[ib*3],navbuf[ib*3+1],
           (int)navbuf[ib*3+2]);
    i=0;
    if (pflg)
      for (k=0; k<NPROC; k++)
        { printf("\n Processor# (%d)  %d\n",k,procnos[k]);
          for (j=0; j< NUMBUF*NWPBa; j++)
            if (s= (sbuf[k*NUMBUF*NWPBa+j] >> 6))
              { printf("%4d %hx   ",j,(s & 0x3ff));
                i++;  if (i > 5) { i=0;  printf("\n");}  }     }
    }
```

```
Dec 20 17:15 1988  navloc.c Page 1

/*     Program Locate Ping
*
*    Input: N x NPROCS (6A) or NPROCS*2 (6B)  array of raw navigation words
*        consisting of the outputs of the one or two 12KHz receivers
*        associated with NPROCS processors
*
*    Output: A 2 channel SIO data file containing the cross-correlation of
*        the simulated transponder ping with the data.  And eventually a
*        display of the bits as they are read in, and a plot of the
*        x-correlation.! */
#include <stdio.h>
#include <math.h>
#include <sys/file.h>
#include "nav.h"

unsigned int mask[30] = {0x20000000,0x10000000,
    0x08000000,0x04000000,0x02000000,0x01000000,
    0x00800000,0x00400000,0x00200000,0x00100000,
    0x00080000,0x00040000,0x00020000,0x00010000,
    0x00008000,0x00004000,0x00002000,0x00001000,
    0x00000800,0x00000400,0x00000200,0x00000100,
    0x00000080,0x00000040,0x00000020,0x00000010,
    0x00000008,0x00000004,0x00000002,0x00000001};

unsigned int nav[64*74/3];
float oave[4] = {0,0,0,0};  int ping = {0},bwinw[4],bwinb[4];
float PINGSIZE[48]={4.,6.,6.,6.,6.,6.,6.,6.,3.,6.,3.,6.,
                     3.,6.,6.,6.,6.,6.,6.,6.,4.,6.,4.,6.,
                     6.,6.,6.,6.,6.,6.,4.,6.,3.,6.,3.,6.,
                     10.,10.,10.,10.,10.,10.,10.,10.,10.,10.,10.,10.};

navloc(proc0,navbuf,start,nrcvr,xpndr,NPROC,pst,bw)
unsigned short proc0[];  int start,nrcvr,xpndr,NPROC,pst[12],*bw;  float navbuf[
{
  int inc,iwdst[12],ibitst[12],itemp[12],ngood,pstart,pend;
  int i,il,jl,k,l,N,q,p,pl,pingend,flag,NAVWORDS,off,ijk,ploff;
  float detec,sum;
  float ave,avenew,timel[12],time2,time3,time,c,temp4;
  unsigned int templ,temp2,temp3;  unsigned short msk= 0x0000FFC0;
  for (k=0;k<12;k++)
    { temp4 = (float)(istart[xpndr][k]+start)/12.;
      iwdst[k]=temp4; ibitst[k]=(int)((temp4 - iwdst[k])*30); }
/* div by 4 = ms/input words, div by 3 words/pkd words = packed words
*   LENGTH = length of valid data window, 250   bits * .4 * 1.5 = about 150
*   meters. The data are received by navloc as the most significant 10 bits
*   in a 16-bit word and repacked into the 30 least significant bits in a
*   32-bit word.  The correlation is done as a moving adder with leading
*   pointers il (word) and jl (bit) and trailing pointers k (word) and
*   l (bit) starting from the beginning of the plot window (bwinw[xpndr]
*   + bwinb[xpndr] the window word and bit pointers) for SAVSZ bits
*   (1 second of data).  The plot window is set up as WINOFF words before
*   the valid data window when the program is initiated and allowed to
*   move as the reply moves by evaluating the average movement of the
*   replies across the array.  The detection is valid only within the valid
*   data window, with parameters pstart, pend, LENGTH; and is defined as a
*   normalized correlation amplitude of detec=1 (normalized by PINGSIZE).
```

```
Dec 20 17:15 1988  navloc.c Page 2

*   The time of the detection is the start of the plot window plus the number
*   of bits, p, as the adder was initilized with the first PINGSIZE outside
*   the correlation loop.   */
  NAVWORDS= NWPBa*(NUMBUF-1); c= .4; off=NWPBa*NUMBUF;
  detec = 1.;
  N = (NAVWORDS/3) * 30;  /*  window parameters */
/*load 32 bit integer array nav with {00 navl nav2 nav3} where navl,nav2,
    nav3 are the 10 bit equally spaced samples of one 12KHz rcvr output. */
for (ijk=0; ijk<nrcvr; ijk++)
  {
  for(i=0; i<N/30; i++)
    { templ = (proc0[ijk*off+i*3]   & msk) << 14;
      temp2 = (proc0[ijk*off+i*3+1] & msk) <<  4;
      temp3 = (proc0[ijk*off+i*3+2] & msk) >>  6;
      nav[i] = templ + temp2 + temp3;           }
/* sum across first ping width of data */
  sum = 0;  time=0;
/*  define beginning of plot window from the bottom receiver for
*   the bottom transponders and from the top receiver for the
*   FLIP transponder */
  if(xpndr != 3){if(ping==0){bwinw[xpndr]=iwdst[0]-WINOFF;bwinb[xpndr]=ibitst[0]
  else {if(ping==0)
            {bwinw[xpndr]=iwdst[nrcvr-1]-WINOFF; bwinb[xpndr]=ibitst[nrcvr-1]
  il=k=bwinw[xpndr]; jl=l=bwinb[xpndr];
  *bw= (bwinw[xpndr]*30 + bwinb[xpndr]) * .4;
  for (pingend=0; pingend<PINGSIZE[xpndr*12+ijk]; pingend++)
    { if (nav[il] & mask[jl])  sum++; jl++; if(jl>=30)  { jl=0; il++; } }
  savsuml[0] = sum;
/* slide ping */
  flag=0;  q=0;
  pstart=(iwdst[ijk]*30+ibitst[ijk])-(k*30+l);
  pend = pstart+LENGTH[xpndr*nrcvr+ijk];
  pst[ijk]=pstart;
  if(pstart<0){printf("\nWindow < Look Area = %d\n",pstart); exit(1); }
  if(pend>SAVSZ){printf("\nWindow > Look Area = %d\n",pend); exit(1); }
  ploff=SAVSZ*(xpndr*nrcvr+ijk);
  for(pl=1+ploff; pl<SAVSZ+ploff; pl++)
      {
      p=pl-ploff;
      if(nav[k] & mask[l++])  sum--;  if(nav[il] & mask[jl++])  sum++;
      savsuml[pl]=sum/PINGSIZE[xpndr*12+ijk];
      if((savsuml[pl] >= detec) && (p >= pstart) && (p <= pend))
        {
        if(flag==0)  {time=bwinw[xpndr]*30 +bwinb[xpndr] + p; flag=1;}
        q++;
    /* printf("sum %d = %d\n",pl,savsuml[pl]); */
        }
      templ = nav[k] & mask[l-1];
      if(l>=30)  { l=0; k++; }  if(jl>=30)  { jl=0; il++; }
      }
  navbuf[ijk*3+2]= q;          /* Size of savsum */
/*change time to milliseconds then to meters*/
/*correct for processor buffering and hardware clock offset */
/* correct for offset at start - One input frame is 5 nav bits */
/* at 0.4 ms/bit = 2ms */
  timel[ijk] = (time * c);
```

```
  time2 = time1[ijk] - start; time3 = time2  - 7 - (128 - .016 * (32-NPROC));
  navbuf[ijk*3]= time3;  navbuf[ijk*3+1]= time3 *1500 /1000;
  if (time3 < 0.) navbuf[ijk*3+1] = 0.;
  }
/* WINDOW perturbation */
ngood=0; ave=0;
for(k=0;k<12;k++)
  { itemp[k] = time1[k] - (istart[xpndr][k]+start);
    if(time1[k] != 0) {  ave += itemp[k];   ngood++; }      }
if(ngood==0){ave=oave[xpndr];} else {ave /= ngood;}
ngood=0;  avenew=0;
for (k=0;k<12;k++)
  { if((itemp[k]>(ave*2))|(itemp[k]<(ave/2))) { itemp[k]=0;}
    else {avenew += itemp[k];  ngood++;}                }
if(ngood!=0) ave=avenew/ngood;
if(ping==0) oave[xpndr]=ave;
if((ping==0) & (xpndr==3)) {ping=1; return;}
inc = (int)(ave - oave[xpndr]);
for(k=0;k<12;k++)   if((abs(inc)) <= 33) istart[xpndr][k] += inc;
printf("iadd=%d\n",inc);
oave[xpndr] = ave;
}
```

Dec 9 20:03 1988 navtst2.c Page 1

```
#include <stdio.h>
#include "nav.h"
unsigned short sbuf[NUMBUF*NWPBb*MAXPROCS*2+NWPBb*2];
fillnavb(start,navbuf,l,fd,NPROC,procnos,rolln,pflg,vflg,nout,navch,fp)
int      start,      l,*fd,NPROC,          rolln,pflg,vflg,     navch;
float navbuf[MAXPROCS*6],nout[];    unsigned char procnos[MAXPROCS];
FILE *fp;
    {
    int i=0,j=0,k=0,joff,jl=0,ib,kout,kin,jout,bufsize,
      nfrerr, tnfrerr,nbadproc,nbadfr;  unsigned char fctr,ofctr;
    unsigned short SYNC= 0xeb90,stemp,stempl,stemp2,oseql,s;
    tnfrerr=0;  bufsize= 16 + 64* (4 + NPROC*16);  oseql = u.sbuf[1]+1;
    printf ("\ncntbuf: %d, start: %d", cntbuf,start);
    printf ("\nhwclock: %d,  SEQN: %d\n",u.sbuf[0],u.sbuf[1]);
    for (i=0; i<NUMBUF; i++) /* Get NUMBUF buffers (>9 sec) for each receiver*/
      {
      nbadfr=nbadproc=0;  ofctr= ((u.cbuf[19] &0x7f) -1)&0x7f;
      for (j=0; j<64; j++)    /* get 64 *2 samples */
        {
        joff=16+j*(4+NPROC*16);    /* to frame */  (++ofctr) ;
        /* frame output pointer [output frames - input frames]  */
        stemp= u.sbuf[joff/2];
        if (stemp != SYNC)
            { fprintf(stderr," Error - Incorrect value of Frame Sync - %x\n",
              stemp); fprintf(stderr," Frame# %d\n",j+1); nbadfr++; break;  }
        fctr = u.cbuf[joff+3] & 0x7f;
        nfrerr= fctr-ofctr;   if(nfrerr < -64) nfrerr += 128;
        if (nfrerr < 0) { printf(
             "Error - Frame counter difference negative (= %d %d)",fctr,ofctr);
             exit(1);    }
        if (nfrerr > 0)
          {
          fprintf(stderr,"Counter incorrect! - %d, should = %d\n",fctr,ofctr);
          ofctr=fctr;
          for (jl=0; jl<nfrerr; jl++)
            for (k=0; k<NPROC; k++)
              { /* kout= output pointer for each processor */
               /* jout= frame output pointer - output frames = input frames */
              kout = k*NUMBUF*NWPBb*2;  if (i > 0) kout+= (i-1)*NWPBb+64;
              jout=(j+jl+tnfrerr);
              sbuf[jout+kout] = 0;  sbuf[jout+kout+NUMBUF*NWPBb] = 0;  }
          tnfrerr += nfrerr;
          }
        for (k=0; k<NPROC; k++)     /* for NPROC Processors */
          {
          jout=(j+tnfrerr);  kout = k*NUMBUF*NWPBb*2 + i*NWPBb;
   /* input pointer for each processor kin */
          kin=k*16+4+15;  stempl= u.sbuf[(joff+kin-3)/2] & 0xffc0;
          stemp2= (*((unsigned short *) &u.cbuf[joff+kin-2])<<4) & 0xffc0;
          if (u.cbuf[joff+kin] != procnos[k])
            { nbadproc++;
              if (k==0) stempl= stemp2= 0;
              else { if (u.cbuf[joff+kin-16] == procnos[k])
                     { stempl= u.sbuf[(joff + kin - 19)/2]  &0xffc0;
                       stemp2= *((unsigned short *)&u.cbuf[joff+kin-18])<<4;
                       stemp2 &= 0xffc0;   }
```

Dec 9 20:03 1988 navtst2.c Page 2

```
                       else stempl= stemp2= 0; }
            }
          sbuf[jout+kout]    = stempl;   sbuf[jout+kout+NUMBUF*NWPBb] = stemp2;
          } /* End of k loop */
        }  /* End of j loop */
      if(nbadfr > 0) fprintf(stderr,"# Bad frame sync's  = %d  Buffer# %d \n"
                     ,nbadfr,cntbuf);
      if(nbadproc > 0) fprintf(stderr,"# Bad processor ID's = %d  Buffer# %d \
                     ,nbadproc,cntbuf);
      cntbuf++;
  /* Note - it is necessary to byte swap the data on the Sun! */
      if (read(*fd,(char *)u.cbuf,bufsize) < bufsize)
          *fd= whichtp(vflg,nout,rolln,NPROC,procnos,navch,*fd,fp);
      swab((char *)u.cbuf,(char *)u.cbuf,bufsize);
      if (u.sbuf[1] == (oseql-1))
          { printf(" Switch tapes; cntbuf= %d\n",cntbuf);
            *fd= whichtp(vflg,nout,rolln,NPROC,procnos,navch,*fd,fp);
            if (read(*fd,(char *)u.cbuf,bufsize) < bufsize)
              *fd= whichtp(vflg,nout,rolln,NPROC,procnos,navch,*fd,fp);
            swab((char *)u.cbuf,(char *)u.cbuf,bufsize);  }
      if (u.sbuf[1] != oseql)
          { fprintf(stderr,
              "Error - incorrect SEQN, should be %d, is %d, Buffer# %d \n",
               oseql,u.sbuf[1],cntbuf);
            for (j=0; j<NWPBb; j++)
              for (k=0; k<NPROC; k++)
                { kout = k*NUMBUF*NWPBb*2 + (i+1)*NWPBb;
                  sbuf[j+kout]= sbuf[j+kout+NUMBUF*NWPBb]= 0;  }
          }
      oseql= u.sbuf[1]+1;
      } /* End of i loop */
  navloc(sbuf,navbuf,start,NPROC*2,l,NPROC);
  if (l == 3)
    for (ib=0; ib<NPROC; ib++)
      { navbuf[ib*6+1] += 89.92; navbuf[ib*6+4] += 89.92; }
  printf ("\nRoll: %d,  Xponder: %d  Start: %d\n",rolln,l,start);
  for (ib=0; ib<NPROC; ib++) printf("Proc# %d;  %f  %f  %d  %f  %f  %d\n",
     ib,navbuf[ib*6],  navbuf[ib*6+1],(int)navbuf[ib*6+2],
        navbuf[ib*6+3],navbuf[ib*6+4],(int)navbuf[ib*6+5]);
  i=0;
  if (pflg)
    for (k=0; k<NPROC; k++)
      { printf("\n Processor# (%d)  %d  (First Navword)\n" ,k,procnos[k]);
        for (j=0; j< NUMBUF*NWPBb; j++)
          if (s= (sbuf[k*NUMBUF*NWPBb*2+j] >> 6))
            { printf("%4d %hx   ",j,(s & 0x3ff));  i++;
              if (i > 5) { i=0; printf("\n");  }         }
        printf("\n Processor# (%d)  %d  (Second Navword)\n",k,procnos[k]);
        for (j=0; j< NUMBUF*NWPBb; j++)
          if (s= (sbuf[k*NUMBUF*NWPBb*2+NUMBUF*NWPBb+j] >> 6))
            { printf("%4d %hx   ",j,(s & 0x3ff));  i++;
              if (i > 5) { i=0;  printf("\n");}         }           }
  }
```

Jul 26 11:58 1988 Makefile Page 1

```
FFLAGS=-g
OBJECTS=flpnav.o dcode.o getoke.o rline.o upcase.o xcor.o \
fix3.o rdline.o linint.o

SOURCES=flpnav.f dcode.f getoke.f rline.f upcase.f xcor.f \
fix3.f rdline.f linint.f

flpnav:$(OBJECTS)
	f77 $(FFLAGS) $(OBJECTS) -o flpnav
```

```
      PROGRAM FLPNAV
c     FLPNAV performs the navigation for R/P FLIP and an array deployed
c     vertically in deep water during September 1987.
c  FLPNAV requires the spatial positions of 3 transponders, and a file with 39
c  slant ranges  one for each of the 12 array receivers to each transponder and
c  one from FLIP to each transponder plus the depths of the array receivers and
c  FLIP. The ouput of FLPNAV are the x,y,z coordinates and rms error for
c  each of the 13 elements.
c
      PARAMETER ( npars = 24 )
C****! the number of user parameters
      PARAMETER ( ntrs = 3 )
C****! the maximum number of transponders allowed
      PARAMETER ( ntrs1 = ntrs+1)
C****! the number of slant ranges per element
      PARAMETER ( nelems = 13 )
C****!the maximum number of things (elements) to fix
      PARAMETER ( maxvel = 1000 )
C****! the maximum number of elements in the vdp array
      PARAMETER ( maxtim = 1000 )
C****! the max length of the Flip slant range file
      PARAMETER ( maxave = 10, maxav = maxave + 1 )
C****!the maximum number of fixes that can be averaged
      DIMENSION trloc(ntrs*4)
C****! x,y,z for ntr transponders
      DIMENSION array(maxav*nelems*ntrsl), sarray(nelems*ntrsl),
     *          tarray(nelems*ntrsl)
      DIMENSION vel(maxvel)
      DIMENSION depth(nelems)
      DIMENSION nzeros(ntrsl)
C****!check for missing xpndr
      DIMENSION ieleno(2)
      DIMENSION red(maxtim), green(maxtim), blue(maxtim)
      DIMENSION timesi(maxtim), timesj(maxtim), timesk(maxtim)
      COMMON /sioln/ cbufin, iichar, nchar, iprint
      CHARACTER*100 cbufin
      DIMENSION deltaz(ntrs), hrange(ntrs)
      DIMENSION trx(maxtim), try(maxtim), trsig(maxtim)
      LOGICAL first
      INTEGER alter, stainc
      CHARACTER*6 names(npars)
C****! allowable parameter names
      DIMENSION lvals(npars), vals(npars)
      EQUIVALENCE ( lvals(1), vals(1) )
      CHARACTER*1 types(npars)
      CHARACTER*80 token, tokenl
      CHARACTER*3 cday
      CHARACTER*40 ctime
      EQUIVALENCE ( ifile, lvals(1) ),
     2            ( depths, vals(2) ),
     3            ( trlocs, vals(3) ),
     4            ( vdp, vals(4) ),
     5            ( xbfile, vals(5) ),
     6            ( oldcv, vals(6) ),
     7            ( trfile, lvals(7) ),
     8            ( lprint, lvals(8) ),
```



```
     9            ( alter, lvals(9) ),
     *            ( thres, vals(10) ),
     1            ( stainc, lvals(11) ),
     2            ( nsta, lvals(12) ),
     3            ( ofile, vals(13) ),
     4            ( redf, vals(14) ),
     5            ( greenf, vals(15) ),
     6            ( bluef, vals(16) ),
     7            ( thresd, vals(17) ),
     8            ( calctd,vals(18) ),
     9            ( lpfile, lvals(19) ),
     *            ( dvp, vals(20) )

      DATA names/ 'IFILE ','DEPTHS','TRLOCS','VDP   ','XBFILE',
     2            'OLDCV ','TRFILE','LPRINT','ALTER ','THRES ',
     3            'STAINC','NSTA  ','OFILE ','REDF  ','GREENF',
     4            'BLUEF ','THRESD','CALCTD','LPFILE','DVP   ',
     5            'N2AVE ','ERFILE','FUDGE ','DVFILE' /
      DATA types/'C',3*'F','C','F','C',2*'L','F',2*'L',4*'C',2*'F','C',
     *          'F','L','C','F','C'/
      DATA first /.TRUE. /, nzeros/0,0,0,0/
      DATA pi/3.14159265358979/, pio2/1.57079632679490/
c****
c****    set the presets
c****
      lun = 11
C****! start the unit number from 11
      iunit = 0
C****! the unit number of ifile
      iounit = 6
C****! use 6 as standard out
      ixunit = 0
C****! the velocity file unit number
      itunit = 0
C****! the transponder file unit number
      idunit = 0
C****! the depth-velocity file unit number
      nvels = 0
C****! the number of elements in the vdp array
      ntrls = 0
C****! the number of elements in the transponder location array
      ndeps = 0
C****! the number of elements in the depths array
      oldcv = 1500.
      lprint = 0
      alter = 3
      thres = 100.
      stainc = 1
      nsta = 1
      iprint = 0
      ndone = 0
      ifunit = 0
      thresd = 5.
      ieleno(1) = 0
      ieleno(2) = 0
      lpunit = 0
```

```
      n2ave = 1
      ierrun = 0
      fudge = 0.
      fday=254.
c****
c****
c****   read the users parameters
c****
   90 CALL rdline
C****! read a command line
      ntokes = 1
  100 CONTINUE
      CALL getoke( token, nchars )
C****! get the next token
      tokenl(1:80) = token(1:nchars)
C****! save the lower case token
      CALL upcase( token, nchars )
C****! convert the token to uppercase
      IF( nchars .LE. 0 ) GOTO 90
      ntokes = ntokes + 1
C****! got a token name (hopefully)
      DO 190 i = 1, npars
         IF( token(1:nchars) .EQ. names(i) ) THEN
             nparam = i
             GOTO 200
         ENDIF
  190 CONTINUE
      IF( token(1:nchars) .EQ. 'END' ) GOTO 1000
      IF( names(nparam) .EQ. 'VDP' ) GOTO 220
C*** ! allow multiple parameter values be given
      IF( names(nparam) .EQ. 'DVP' ) GOTO 220
C***! allow multiple parameter values be given
      IF( names(nparam) .EQ. 'TRLOCS' ) GOTO 220
C***! allow multiple parameter values be given
      IF( names(nparam) .EQ. 'DEPTHS' ) GOTO 220
C***! allow multiple parameter values be given
      IF( names(nparam) .EQ. 'CALCTD' ) GOTO 220
C***! allow multiple parameter values be given
      PRINT *,' ***  ERROR  ***  FLPNAV does not have a parameter',
     *     token(1:nchars)
      ierror = ierror + 1
      GOTO 100
c****
c****   Now get the value of the parameter
c****
  200 CONTINUE
      CALL getoke( token, nchars )
      tokenl(1:80) = token(1:nchars)
C****! save the lower case token
      IF( nchars .LE. 0 ) THEN
          CALL rdline
C****! read a command line
          ntokes = 1
          GOTO 200
      ENDIF
      ntokes = ntokes + 1
```



```
      IF( names(nparam) .EQ. 'IFILE ' ) THEN
          iunit = lun
          OPEN( UNIT = iunit,
     *           FILE = tokenl,
     *           STATUS = 'OLD',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'OFILE ' ) THEN
          iounit = lun
          OPEN( UNIT = iounit,
     *           FILE = tokenl,
     *           STATUS = 'UNKNOWN',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'LPFILE ' ) THEN
          lpunit = lun
          OPEN( UNIT = lpunit,
     *           FILE = tokenl,
     *           STATUS = 'UNKNOWN',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'REDF' ) THEN
          ifunit = lun
          OPEN( UNIT = ifunit,
     *           FILE = tokenl,
     *           STATUS = 'UNKNOWN',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'GREENF' ) THEN
          jfunit = lun
          OPEN( UNIT = jfunit,
     *           FILE = tokenl,
     *           STATUS = 'UNKNOWN',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'BLUEF' ) THEN
          kfunit = lun
          OPEN( UNIT = kfunit,
     *           FILE = tokenl,
     *           STATUS = 'UNKNOWN',
     *           FORM = 'FORMATTED')
          lun = lun + 1
          GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'XBFILE ' ) THEN
          ixunit = lun
```

```
      OPEN( UNIT = ixunit,
     *       FILE = tokenl,
     *       STATUS = 'OLD',
     *       FORM = 'FORMATTED')
      lun = lun + 1
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'TRFILE' ) THEN
      itunit = lun
      OPEN( UNIT = itunit,
     *       FILE = tokenl,
     *       STATUS = 'OLD',
     *       FORM = 'FORMATTED')
      lun = lun + 1
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'DVFILE' ) THEN
      idunit = lun
      OPEN( UNIT = idunit,
     *       FILE = tokenl,
     *       STATUS = 'OLD',
     *       FORM = 'FORMATTED')
      lun = lun + 1
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'ERFILE' ) THEN
      ierrun = lun
      OPEN( UNIT = ierrun,
     *       FILE = tokenl,
     *       STATUS = 'UNKNOWN',
     *       FORM = 'FORMATTED')
      lun = lun + 1
      GOTO 100
   ENDIF
 220 CALL dcode( token, nchars, value, istat )
C***! decode the value (convert to binary!)
   IF( istat .NE. 2 ) ierror = ierror + 1
   IF( names(nparam) .EQ. 'VDP' ) THEN
      nvels = nvels + 1
      vel(nvels) = value
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'DVP' ) THEN
      nvels = nvels + 1
      iswap = (nvels - nvels/2*2)*2-1
      vel(nvels+iswap) = value
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'TRLOCS' ) THEN
      ntrls = ntrls + 1
      trloc(ntrls) = value
      GOTO 100
   ENDIF
   IF( names(nparam) .EQ. 'DEPTHS' ) THEN
      ndeps = ndeps + 1
      depth(ndeps) = value
```



```
         GOTO 100
      ENDIF
      IF( names(nparam) .EQ. 'CALCTD' ) THEN
         IF( ieleno(1) .EQ. 0 ) THEN
            ieleno(1) = value
         ELSE
            ieleno(2) = value
         ENDIF
         GOTO 100
      ENDIF
c**** read in all other parameters which are not explicitly named
      IF( types(nparam) .EQ. 'F' ) THEN
         vals(nparam) = value
      ELSE
         lvals(nparam) = value
      ENDIF
      GOTO 100
c****
c****
c****    Now start to do the real work, we got all of the user's parameters!
c****
c****
 1000 CONTINUE
      IF( n2ave .LE. 0 .OR. n2ave .GT. maxave ) THEN
         PRINT *,' ***  ERROR  ***  N2AVE must be between 0 and ',
     *          max2av
         ierror = ierror + 1
      ENDIF
      PRINT *,ierror,' errors in the job.'
      IF( ierror .GT. 0 ) CALL EXIT
c****
c****   Get the transponders taken care of - adjust the depths for velocity.
c****
      IF( itunit .NE. 0 ) THEN
C****! are they on disk?
         k = 1
C****! count the entries as we read them
         READ( itunit, '(A1)' ) token
C****! the first line is blank
         DO 1010 j = 1, 3
C****! x, y, z of each transponder
            READ( itunit, '(10x,F6.0,2x,F6.0,1x,F(.0,1x,F6.3)' )
     *            (trloc(i),i=k,k+3)
            print *,'trloc',(trloc(i),i=k,k+3)
            k = k + 4
C XXXXX
 1010    CONTINUE
      ENDIF
c**** Get the depth,sound velocity pairs from a file
c**** swap the order (ie. depth,velocity -> velocity,depth)
c**** XXXX
      IF (idunit .NE. 0) THEN
 1100    icnt = 1
C****! count the entries as we read them
 1110    CALL rline( idunit )
         CALL getoke( token, nchars)
```

```
C****! parse the line
          IF( nchars .NE. 0 ) THEN
              iswap = (icnt - icnt/2*2)*2 - 1
C****! was there a token?
              CALL dcode( token, nchars, vel(icnt+iswap), istat )
C****! decode the token and put in time
              icnt = icnt + 1
              iswap = (icnt - icnt/2*2)*2 - 1
              CALL getoke( token, nchars )
              CALL dcode( token, nchars, vel(icnt+iswap), istat )
              icnt = icnt + 1
              GOTO 1110
          ENDIF
1112          print *,icnt-1
          nvels = icnt-1
          DO 1120 jjj=1,icnt-1,2
1120         print *,vel(jjj),vel(jjj+1)
      ENDIF
      IF( iunit .EQ. 0 ) THEN
          iunit = lun
          OPEN( UNIT = iunit,
     *          FILE = 'rnglog.dat',
     *          STATUS = 'OLD',
     *          FORM = 'FORMATTED' )
          lun = lun + 1
      ENDIF
c****
c****   Get all of Flip's data and set up for a qubic spline interpolation
c****  according to time.
c****
      IF( ifunit .NE. 0 ) THEN
C****! is it there?
 1200     icount = 1
C****! count the entries as we read them
 1210     CALL rline( ifunit )
          CALL getoke( token, nchars)
C****! parse the line
          IF( nchars .NE. 0 ) THEN
C****! was there a token?
              CALL dcode( token, nchars, timesi(icount), istat )
C****! decode the token and put in time
              CALL getoke( token, nchars )
              CALL dcode( token, nchars, red(icount), istat )
              icount = icount + 1
              GOTO 1210
          ENDIF
          jcount = 1
 1230     CALL rline( jfunit )
          CALL getoke( token, nchars)
C****! parse the line
          IF( nchars .NE. 0 ) THEN
C****! was there a token?
              CALL dcode( token, nchars, timesj(jcount), istat )
C***! decode the token and put in time
              CALL getoke( token, nchars )
              CALL dcode( token, nchars, green(jcount), istat )
```



```
              jcount = jcount + 1
              GOTO 1230
          ENDIF
          kcount = 1
 1250     CALL rline( kfunit )
          CALL getoke( token, nchars)
C****! parse the line
          IF( nchars .NE. 0 ) THEN
C****! was there a token?
              CALL dcode( token, nchars, timesk(kcount), istat )
C***! decode the token and put in time
              CALL getoke( token, nchars )
              CALL dcode( token, nchars, blue(kcount), istat )
              kcount = kcount + 1
              GOTO 1250
          ENDIF
      ENDIF
c****
c****   Get a set of slant ranges, and depths,  consisting of a slant range
c****  to each transponder from Flip and the array followed by the depth of
c****  the thing being fixed.
c****   The first 3 slant ranges are Flip's, so the are one-way slant ranges.
c****   The array's slant ranges where converted from two-way travel times
c****  (Flip to transponder to receiver) by multiplying by a constant velocity
c****  (no one knows what velocity was used or even if it was a constant) and
c****  dividing by 2. in order to get a distance from the receiver to element.
c****     If this assumption is wrong, then this navigation is wrong!
c****     The first 2 things for every day/time location fix is the
c****  Julian day and the GMT of the location we are fixing!
c****
 2000 CONTINUE
      iprint = lprint
C****! should rline print the data?
      j = 1
      jj = 1
      CALL rline( iunit )
      IF( nchar .EQ. 0 ) GOTO 5000
C****! any more fixes?
      CALL getoke( token, nchars)
C****! parse the line
      IF( nchars .EQ. 0 ) GOTO 2010
C****! was there a token?
      cday = token(1:nchars)
C****! the julian day in characters!
      READ( token, '(I3)' ) iday
      CALL getoke( token, nchars)
C****! parse the line
      ctime = token(1:nchars)
      IF( token(2:2) .EQ. ':' ) THEN
          READ (token,'(I1,1x,I2,1x,F6.3)' ) ihour, imin, sec
      ELSE
          READ (token,'(I2,1x,I2,1x,F6.3)' ) ihour, imin, sec
      ENDIF
      rmin = imin + sec/60.
      GOTO 2040
 2010 CONTINUE
```

```
      CALL rline( iunit )
      IF( nchar .EQ. 0 ) GOTO 5000
 2040 CALL getoke( token, nchars)
C****! parse the line
      IF( nchars .EQ. 0 ) GOTO 2010
C****! was there a token?
      CALL dcode( token, nchars, sarray(j), istat )
C***! decode the token and put the value away
      IF( sarray(j) .LE. 0. .OR. sarray(j) .EQ. 89.92 )
     *     nzeros(jj) = nzeros(jj) + 1
      IF( first ) THEN
          array(j) = sarray(j)
          tarray(j) = sarray(j)
      ENDIF
      IF( j .LT. nelems*ntrsl ) THEN
          j = j + 1
          jj = jj + 1
          IF (jj .EQ. (ntrsl+1)) jj = 1
          GOTO 2040
      ENDIF
      first = .FALSE.
c****
c****  If too many were zero, the navigation must have been turned off, which
c****  was done every hour on the hour for 5 or so minutes, just forget it and
c****  don't even output it!
c****
      nbadfix = 0
      DO 2042 jj = 1, ntrsl
         IF(nzeros(jj) .GT. 7) nbadfix = 1
 2042 CONTINUE
      IF( nbadfix .EQ. 1 ) THEN
          IF( ierrun .NE. 0 ) THEN
              WRITE( ierrun,2041) cday,ctime
 2041         FORMAT(1x,A3,1x,A20,'is a bad fix.')
          ENDIF
          GOTO 4800
      ENDIF
c****
c****  If FFILE was given, then use Flip's slant ranges from it rather than
c****  the the original ifile.  Actually, we already read them and now we just
c****  to find the interpolated value for this time.
c****  Use linear interpolation due to the nondifferentiable qualities of
c****  the Flip file.  Change time to number of minutes from day 245 (fday).
c****
      IF( ifunit .NE. 0 ) THEN
          time = (iday-fday)*1440. + (ihour)*60. + rmin
c***          print *,time,sarray(1),sarray(2),sarray(3)
          CALL linint( timesi, red, icount, time, sarray(1) )
          CALL linint( timesj, green, jcount, time, sarray(2) )
          CALL linint( timesk, blue, kcount, time, sarray(3) )
          ftime = iday + ihour/24. + (imin*60 + sec)/86400.
          print *,ftime,sarray(1),sarray(2),sarray(3)
      ENDIF
c****
c****  Check for bad slant ranges.  ARRAY contains 3 ranges and a depth
c****  for each of the "elements" (Flip and 12 hydrophones).
```



```
c****
         k = 1
         DO 2080 i = 1, nelems
            DO 2070 j = 1, ntrs
               IF( alter .EQ. 0 ) THEN
                  array(k) = sarray(k)
                  GOTO 2065
               ENDIF
               IF( alter .LT. 3 ) THEN
                  array(k) = tarray(k)
                  IF( ABS(tarray(k)-sarray(k)) .LT. thres ) THEN
                     array(k) = sarray(k)
                  ELSE
                     IF( alter .EQ. 1 ) array(k) = 0.
                     IF( alter .EQ. 2 ) array(k) = tarray(k)
                  ENDIF
               ENDIF
C****! leave it alone if alter=2
               IF( alter .EQ. 3 ) THEN
                  IF( i .EQ. 1 ) THEN
C****! can't compare Flip to anything
                     array(k) = sarray(k)
                     GOTO 2065
                  ENDIF
                  IF( ABS(sarray(k)-tarray(k)) .LT. thres ) THEN
                     array(k) = sarray(k)
                     GOTO 2065
                  ENDIF
                  IF( sarray(k+ntrsl) .EQ. 0 .OR.
     *               ABS(sarray(k+ntrsl)-tarray(k+ntrsl)) .GE. thres ) THEN
                     array(k) = tarray(k)
                     GOTO 2065
                  ENDIF
                  array(k) = (array(k-ntrsl) + sarray(k+ntrsl))/2.
C***    ! interpolate between adjacent elements on this station
               ENDIF
               IF( alter .EQ. 4 ) THEN
                  IF( i .EQ. 1 ) THEN
C****! is it Flip?
                     array(k) = sarray(k)
                     GOTO 2065
                  ENDIF
                  IF( ABS(sarray(k)-tarray(k)) .LT. thres .AND.
     *               sarray(k) .GT. 0. ) THEN
                     array(k) = sarray(k)
                     GOTO 2065
                  ENDIF
                  IF( i .EQ. 2 ) THEN
                     ifirst = 0
                     lindex = (nelems-2)*ntrsl
                     DO 2050 l = ntrsl, lindex, ntrsl
C****! find 2 good ranges
                        IF( sarray(k+l) .GT. 0. ) THEN
                           IF( ifirst .EQ. 0 ) THEN
                              nfirst = k + 1
C****! save the index
```

```
                                 ifirst = sarray(nfirst)
C****! save the slant range
                              ELSE
C****! extrapolate
                                 n = (K+1-nfirst)/ntrsl
C***! the number of elements between the good slant ranges
                                 array(k) = ifirst-(sarray(k+1)-ifirst)/n*
     *                             ((nfirst-k)/ntrsl)
                                 GOTO 2065
                              ENDIF
                           ENDIF
 2050                   CONTINUE
                        IF( ierrun .NE. 0 ) THEN
                           WRITE( ierrun, 2051 ) cday,ctime,j
 2051                    FORMAT(1x,A3,1x,A20,' transponder ',I2,' is bad.',
     *                       ' Fix ignored.')
                           GOTO 4800
                        ENDIF
                        GOTO 2065
                     ENDIF
                     IF( i .EQ. nelems ) THEN
                        array(k) = array(k-ntrsl) +
     *                     array(k-ntrsl) - array(k-ntrsl*2)
C****! extrapolate
C***- we are guaranteed the last 2 are good
                        GOTO 2065
C****! use the range for the last station
                     ENDIF
                     lindex = (nelems-i) * ntrsl
C    ! the number of elements afterwards
                     DO 2060 l = ntrsl, lindex, ntrsl
                        IF( sarray(k+l) .NE. 0. ) THEN
                           nfirst = k - ntrsl
C***! the element before is guaranteed to be good
                           ifirst = array(nfirst)
                           n = (k + l - nfirst) / ntrsl
                           array(k) = ifirst + (sarray(k+l)-ifirst) / n
                           GOTO 2065
                        ENDIF
 2060                CONTINUE
                     array(k) = array(k-ntrsl) +
     *                     array(k-ntrsl) - array(k-ntrsl*2)
C***           ! extrapolate - we are guaranteed the last 2 are good
                     GOTO 2065
                  ENDIF
 2065             CONTINUE
                  tarray(k) = array(k)
                  k = k + 1
 2070          CONTINUE
               IF( i .EQ. 1 .OR. alter .EQ. 0 ) THEN
                  array(k) = sarray(k)
                  GOTO 2079
               ENDIF
               IF( ABS(sarray(k)-tarray(k)) .LE. thresd
     *              .AND. sarray(k) .GT. 89.92 ) THEN
                  array(k) = sarray(k)
```



```
               ELSE
                  IF( ierrun .NE. 0 ) WRITE(ierrun,2071) cday,ctime,i
 2071             FORMAT(1x,A3,1x,A30,' depth ',I2,' is bad.')
                  IF( alter .NE. 4 ) THEN
                     array(k) = tarray(k)
                  ELSE
                     IF( i .EQ. 2 ) THEN
                        ifirst = 0
                        lindex = (nelems-2)*ntrsl
                        DO 2075 l = ntrsl, lindex, ntrsl
C****! find 2 good depths
                           IF( sarray(k+l) .GT. 89.92 ) THEN
                              IF( ifirst .EQ. 0 ) THEN
                                 nfirst = k + 1
C****! save the index
                                 ifirst = sarray(nfirst)
C****! save the slant range
                              ELSE
C****! extrapolate
                                 n = (K+l-nfirst+1)/ntrsl
C***      ! the number of elements between the good slant ranges
                                 array(k) = ifirst-(sarray(k+l)-ifirst)/n*
     *                             ((nfirst-ntrsl*2)/ntrsl)
                                 GOTO 2079
                              ENDIF
                           ENDIF
 2075                   CONTINUE
                        IF( ierrun .NE. 0 ) THEN
                           WRITE( ierrun, 2076 ) cday,ctime
 2076                    FORMAT(1x,A3,1x,A20,' depths are bad. ',
     *                       ' Fix ignored.')
                           GOTO 4800
                        ENDIF
                        GOTO 2079
                     ENDIF
                     IF( i .EQ. nelems ) THEN
                        array(k) = array(k-ntrsl) +
     *                     array(k-ntrsl) - array(k-ntrsl*2)
C***      ! extrapolate - we are guaranteed the last 2 are good
                        GOTO 2079
C****! use the range for the last station
                     ENDIF
                     lindex = (nelems-i) * ntrsl
C****! the number of elements afterwards
                     DO 2078 l = ntrsl, lindex, ntrsl
                        IF( sarray(k+l) .GT. 89.92 ) THEN
                           nfirst = k - ntrsl
C***! the element before is guaranteed to be good
                           ifirst = array(nfirst)
                           n = (k + l - nfirst) / ntrsl
                           array(k) = ifirst + (sarray(k+l)-ifirst) / n
                           GOTO 2079
                        ENDIF
 2078                CONTINUE
                     array(k) = array(k-ntrsl) +
     *                     array(k-ntrsl) - array(k-ntrsl*2)
```

```
C*** ! extrapolate - we are guaranteed the last 2 are good
                GOTO 2079
            ENDIF
         ENDIF
 2079    tarray(k) = array(k)
         k = k+ 1
 2080 CONTINUE
c****
c****  if the user gave depths, put them over that from the data file
c****
      IF( ndeps .NE. 0 ) THEN
          DO 2090 i = 1, nelems
             array(i*4) = depth(i)
 2090     CONTINUE
      ENDIF
c****
c****  Adjust the element depths for the velocity correction - Flip's
c****  depth was measured with a ruler, so don't correct it!  The
c****  element depth is measured from 0. (harrynav converts the slant
c****  range from the bottom of flip to the element and then adds in
c****  the constant depth of the transponder on flip).  Therfore, subtract
c****  the depth of flip's transponder from the element depth before
c****  doing the velocity correction - remember that the the slant range
c****  started at Flip's depth, not 0.!  Then add flip's depth back,
c****  because depth is measured from sea level!
c****
      IF( nvels .NE  0 ) THEN
          DO 2100 i = 2, nelems
             array(i*4) = array(i*4) - array(4)
             CALL xcor(array(i*4),array(4),oldcv,vel,nvels,array(i*4),
     *       array(4) )
             array(i*4) = array(i*4) + array(4)
 2100     CONTINUE
      ENDIF
c****
c****    Subtract the uncorrected leg 1 (flip-transponder) from the
c****  two way slant ranges, leaving leg 2 (transponder-array)(uncorrected)
c****
      k = 5
      DO 2200 i = 2, nelems
         DO 2190 j = 1, ntrs
            array(k) = array(k) - array(j)
            k = k + 1
 2190    CONTINUE
         k = k + 1
 2200 CONTINUE
c****
c****  Add in FUDGE to the slant ranges
c****
      IF( fudge .NE. 0.) THEN
          k = 5
          DO 2140 j = 2, nelems
             DO 2150 i = 1, ntrs
                array(k) = array(k) + fudge
                k = k + 1
 2150        CONTINUE
```



```
                k = k + 1
 2140     CONTINUE
      ENDIF
c****
c****    Adjust Flip's slant ranges for velocity
c****
      DO 2300 i = 1, ntrs
         IF( nvels .NE. 0 )
     *      CALL xcor( array(i), array(4), oldcv, vel, nvels, array(i),
     *      trloc((i-1)*4+3) )
 2300 CONTINUE
c****
c****   Save this set of slant ranges and average them
c****
      IF( n2ave .GT. 1 ) THEN
          n2move = maxave * (ntrs+1) * nelems
          last = maxav * (ntrs+1) * nelems
          inc = (ntrs+1) * nelems
          DO 2410 i = 1, n2move
 2410     array(last-i+1) = array(n2move-i+1)
          DO 2420 i = 1, inc
 2420     array(i) = 0.
          n2do = n2ave
          IF( ndone .LT. n2ave ) n2do = ndone + 1
          DO 2440 i = 1, n2do
             DO 2430 j = 1, inc
                array(j) = array(j) + array(j+i*inc)
 2430        CONTINUE
 2440     CONTINUE
          temp = FLOAT(n2do)
          DO 2450 i = 1, inc
 2450     array(i) = array(i) / temp
      ENDIF
c****
c****    Adjust the array slant ranges for velocity
c****
      IF( nvels .NE. 0 ) THEN
          k = 5
          DO 2600 i = 2, nelems
             DO 2590 j = 1, ntrs
                CALL xcor( array(k), array(i*4), oldcv, vel, nvels,
     *           array(k), trloc((j-1)*4+3) )
                k = k + 1
 2590        CONTINUE
             k = k + 1
 2600     CONTINUE
      ENDIF
c****
c****   Figure out new tranponder depths
c****
      IF( ieleno(1) .NE. 0 ) THEN
          itemp1 = ieleno(1)*4 + 4
          itemp2 = ieleno(2)*4 + 4
          a = array(itemp1) - array(itemp2)
          DO 2700 i = 1, ntrs
             b = array(ieleno(1)*4+i)
```

```
            c = array(ieleno(2)*4+i)
            beta = acos((-b*b+a*a+c*c)/(2*a*c))
            e = sin(pio2-beta) * c
            trdep = array(ieleno(2)*4+4) + e
 2700    CONTINUE
      ENDIF
c****
c****  Write the one way slant ranges (from the transponder to the
c****  elements) after velocity correction.  The depth has also been
c****  corrected for velocity.
c****  This file can be used in the looping program XPMAIN.
c****
      IF( lpunit .NE. 0 ) THEN
         DO 2800 i = 1, nelems-1
            idepth = array(i*4+4)
            ired = array(i*4+1)
            igreen = array(i*4+2)
            iblue = array(i*4+3)
            WRITE( lpunit, 2790 ) iday, ihour, imin, idepth,
     *         ired, igreen, iblue
 2790       FORMAT(I3,'/',2I2,7x,'0','  2400.  3200.',3x,'0',I5,
     *          4x,'0',3I6)
 2800    CONTINUE
      ENDIF
c****
c****  Print the slant ranges?
c****
      IF( AND(lprint,1) .NE. 0 ) THEN
         DO 2900 i = 1, nelems
            PRINT *, (array(j),j=(i-1)*4+1 (i-1)*4+4)
 2900    CONTINUE
      ENDIF
c****
c****  Now calculate a FLIP FIX - deltaz is the vertical distance between
c**** Flip and the transponder.  hrange is the horizontal distance between
c**** Flip and each transponder.
c****
      WRITE (iounit,'(1x,A3,1x,A20)' ) cday,ctime
      ngood = 0
C****! count the good fixes
      DO 3000 i = 1, ntrs
         IF( array(i) .LE. 0 ) GOTO 3050
C****! don't fix bad data
         j = (i-1) * 4
         trx(i) = trloc(j+1)
         try(i) = trloc(j+2)
         trsig(i)= trloc(j+4)
         deltaz(i) = ABS( array(4) - trloc(j+3) )
         hrange(i) = SQRT(array(i)*array(i)-deltaz(i)*deltaz(i))
         IF( AND(lprint,1) .NE. 0 )
     *      PRINT *,i,trx(i),try(i),deltaz(i),hrange(i)
 3000 CONTINUE
C****  For FLIP initial position, use closest GPS position.
C****  Since FLIP did not move substantially, a constant position is used.(2400,
C****  If GPS positions are not available then call trgss to estimate initial
c****  geometrically.
```



```
c****      CALL trgss( hrange, ntrs, trx, try, x, y, err )
      x=2400.
      y=3200.
      CALL fix3( hrange, trx, try, trsig, x, y, err )
      WRITE (iounit,'(4(3x,f9.4))') x,y,array(4),err
      ngood = 1
c****
c****  Fix the array - use Flip's position as the guess for the first
c****  element, then use the previous array element's fix as the guess for
c****  successive elements.
c****
 3050 CONTINUE
      k = 5
      DO 4000 i = 2, nelems
c****    don't do a fix on bad data
         IF( array(k) .LE. 0. .OR. array(k+1) .LE. 0. .OR.
     *       array(k+2) .LE. 0. ) THEN
             IF( ierrun .NE. 0 ) WRITE( ierrun, 3060) cday,ctime,k
 3060        FORMAT(1x,A3,1x,A20,' index ',I2,' has a bad value.')
             k = k + 4
             GOTO 4000
         ELSE
            ngood = ngood + 1
         ENDIF
         DO 3100 j = 1, ntrs
            l = (j-1) * 4
            trx(j) = trloc(l+1)
            try(j) = trloc(l+2)
            trsig(j) = trloc(l+4)
            deltaz(j) = ABS( array(i*4) - trloc(l+3) )
            hrange(j) = SQRT(array(k)*array(k)-deltaz(j)*deltaz(j))
            IF( AND(lprint,1) .NE. 0 )
     *         PRINT *,j,trx(j),try(j),deltaz(j),hrange(j),array(k)
            k = k + 1
 3100    CONTINUE
         k = k + 1
C*     For array initial positions, use previous position of FLIP or array eleme
c****  Alternately, use a known GPS positions (2400,3200), as a last resort,
C****  call trgss to estimate initial geometrically.
C****         CALL trgss( hrange, ntrs, trx, try, x, y, err )
c          x=2400.
c          y=3200.
         CALL fix3( hrange, trx, try, trsig, x, y, err )
         WRITE (iounit,'(4(3x,f9.4))') x,y,array(i*4),err
 4000 CONTINUE
      IF( ngood .NE. nelems ) THEN
          DO 4100 i = 1, nelems-ngood
             PRINT*,' BAD Element fix: x=',x,' y=',y,' err=',err
 4100     CONTINUE
      ENDIF
c
 4800 CONTINUE
      ndone = ndone + 1
      DO 4804 jj = 1,ntrsl
 4804 nzeros(jj) = 0
      IF( ndone .EQ. nsta ) GOTO 5000
```

```
      IF( stainc .LE. 1 ) GOTO 2000
C****! do the next station?
      DO 4900 i = 1, stainc-1
C****! skip stainc-1 stations
         DO 4890 j = 1, nelems
C****! skip 1 line for every element
            CALL rline( iunit )
C****! read a line from IFILE
            IF( nchar .EQ. 0 ) GOTO 5000
 4890    CONTINUE
 4900 CONTINUE
      GOTO 2000
C****! do the next station
c
 5000 CONTINUE
      END
```

Jul 26 11:35 1988 fix3.f Page 1

```
      SUBROUTINE FIX3 (RANGE,TRX,TRY,TRSIG, X,Y,ERR)
c  FIX3 iterates the xy array/flip locations
c
c ARGUMENTS:
c hrange - The array of 3 transponder horizontal distances (the horizontal
c          or projected part of the slant range).
c trx    - The 3 x-coordinates of the 3 transponders.
c try    - The 3 y-coordinates of the 3 transponders.
c x      - Initially x is the estimated array/flip x coordinate.
c           FIX3 iterates the x coordinate minimizing the RMS error.
c y      - Initially y is the estimated array/flip y coordinate.
c           FIX3 iterates the y coordinate minimizing the RMS error.
c err    - An error in the fix location.
c         >0.,  The average error squared of the fix.
c         <0.,  The average error, but "TOO MANY LOOPS".
c
      REAL    RANGE(6), TRX(6), TRY(6), TRSIG(6)
      SAVE errold
      DATA    HUGE /1.6E38/,LOOPS /50/,ERRMIN /.25/,RE /.01/,GAIN /1.5/
     *        ZERO /.1/
     DATA errold/0./                                               ! preset
c
      ISIGN = 1
      ERROLD = HUGE
c                               !set previous error squared to a huge number.
      DO 200 I=1,LOOPS
          DX = 0.0
c                               !initialize correction vector components
          DY = 0.0
          ERR = 0.0
c                               !initialize cumulative error squared sum.
          COUNT = 0.0
          sig=1.0
          DO 100 J=1,3
              XDIFF = X - TRX(j)
c                               !calculate horiz range from
              YDIFF = Y - TRY(j)
c                               !fix to transponder...
              RCALC = SQRT(XDIFF*XDIFF+YDIFF*YDIFF)
              IF( RCALC .GT. ZERO ) THEN
                  EVCTR = (RANGE(j) - RCALC)/TRSIG(j)
c                               !diff between calc
                  ERR = ERR + EVCTR*EVCTR
c                               ! and observed ranges.
                  RATIO = EVCTR/RCALC
                  DX = DX + RATIO*XDIFF
c                               !sum X-component of diff vectr
                  DY = DY + RATIO*YDIFF
c                               !sum Y-component of diff vectr
                  COUNT = COUNT + 1.0
              ENDIF
 100          CONTINUE
c
          ERR = ERR/COUNT
c                               !average error squared of fix
c       ...if error is small enough or is no longer converging...quit
```

Jul 26 11:35 1988 fix3.f Page 2

```
          IF((ERR.LT.ERRMIN) .OR. ((ERROLD-ERR).LT.RE)) GOTO 300
          ERROLD = ERR
c                               !save for comparison with next err
          X = X + GAIN * DX/COUNT
c                               !add correction vector components
          Y = Y + GAIN * DY/COUNT
 200      CONTINUE
      ISIGN = -1
c                               !flag too many loops
c
 300  CONTINUE
      ERR = SQRT(ERR) * ISIGN
c                               !flag too many loops with neg. error
      RETURN
      END
```

Jul 26 09:53 1988 trgss.f Page 1

```
      SUBROUTINE TRGSS(HRANGE,NHORIZ,TRX,TRY,XX,YY,ERR)
C
C     TRGSS calls FIX2 to calculate two possible "fix" positions,
C     on the basis of ranges from 2 transponders.  One of the two
C     positions is selected according to the following criteria;
C       1) If a range to a third transponder exists, select the
C          position whose distance from that transponder most nearly
C          corresponds to the observed range.  Otherwise...
C       2) Choose the position nearest to the predicted position for
C          this fix.  At present, the predicted position is simply
C          the previous position for this device.  If the times as
C          well as positions of previous fixes were more readily
C          avaliable, previous positions and times could be used to
C          make better predictions.
C          The value off xx and yy will be used as the prediction.
C          On return, the new position is returned in xx and yy.
C
C ARGUMENTS:
C hrange - The array of 3 horizontal ranges (projected slant ranges) to the
C          3 transponders from the thing being fixed.  This is the "one-way"
C          range!
C nhoriz - the number of transponders.  The number of hranges and trx and try.
C trx    - The array of up to 3 x coordinates of the transponders.
C try    - The array of up to 3 y coordinates of the transponders.
C xx     - The x coordinate of the array/flip fix.  Initially this is a guess o
C              expected location. This coordinate is then iterated by fix3.
C yy     - The y coordinate of the array/flip fix.  Initially this is a guess o
C              expected location. This coordinate is then iterated by fix3.
C err    - A pass through argument to routine FIX2.  ERR is not modified.
C
      REAL HRANGE(3),  X(2), Y(2), TRX(3), TRY(3), XX, YY
C
      DISTSQ(Xz,Yz) = (Xz*Xz+Yz*Yz)
C
      I=2
      IF( nhoriz .LE. 1 ) RETURN
      ERR = 0.0
C      ...Find the two possible positions (X1,Y1) AND (X2,Y2).
      CALL FIX2 (HRANGE,TRX,TRY,X,Y,ERR)
C
C      ...If a 3rd horizontal range is known, choose the position whose
C         distance from the transponder is closest to that range,
C         otherwise, choose the position that is closest to an
C         estimated position.
C
    IF( NHORIZ .EQ. 3 )THEN
        X3 = TRX(2)
        Y3 = TRY(2)
        R3SQ = HRANGE(2)*HRANGE(2)
        D1 = ABS(DISTSQ(X(1)-X3,Y(1)-Y3)-R3SQ)
        D2 = ABS(DISTSQ(X(2)-X3,Y(2)-Y3)-R3SQ)
    ELSE
C      ...Make some estimate of the position here.  This actually
C         should be projected from the last known position, but at
C         present is just the last position.  Some time info that
C         is required for this is not readily available.
```

Jul 26 09:53 1988 trgss.f Page 2

```
C
            D1 = DISTSQ(X(1)-XX,Y(1)-YY)
C                                            !distance from predicted
            D2 = DISTSQ(X(2)-XX,Y(2)-YY)
C                                            ! position to each candidate
      ENDIF
C
      IF(D1.LT.D2)  I = 1
      XX = X(I)
      YY = Y(I)
      RETURN
        END
```

```
      SUBROUTINE LININT(XA,YA,N,X,Y)
      DIMENSION XA(N),YA(N)
      KLO=1
      KHI=N
 1    IF (KHI-KLO.GT.1) THEN
               K=(KHI+KLO)/2
               IF XA(K).GT.X) THEN
                       KHI=K
               ELSE
                       KLO=K
               ENDIF
        GOTO 1
        ENDIF
        H=XA(KHI)-XA(KLO)
        G=YA(KHI)-YA(KLO)
        IF (H.EQ.0.) PAUSE 'BAD XA INPUT'
        B=(X-XA(KLO))/H
        Y= (B*G)+YA(KLO)
C       PRINT *,XA(KHI),XA(KLO),YA(KHI),YA(KLO)
C       PRINT *,H,G,B,Y
       RETURN
       END
```

```
Dec 21 19:51 1988  fix2.f Page 1

      SUBROUTINE FIX2(RANGE,TRX,TRY,X,Y,ERR)
C
C     FIX2 calculates the initial xy positions from the horizontal ranges of
C       two of the transponders
C     USE RED and BLUE
C
C ARGUMENTS:
C hrange - the horizontal or projected ranges (2 of them).  The horizontal
C          range is the horizontal part of the slant range.
C trx    - the x coordinates of the two transponders.
C try    - the y coordinates of the two transponders.
C x      - An array of 2 possible a coordinates of the thing being fixed.
C          This array is returned by fix2.
C y      - An array of 2 possible a coordinates of the thing being fixed.
C          This array is returned by fix2.
C err      set to 700 by fix2 if " neg. non-intersecting arcs"
C
      REAL RANGE(3), X(2), Y(2), TRX(3), TRY(3)
C
C
      ERR = 0.
C                                                  !number of xpndr for R(2)
      R1 = RANGE(1)
C                                                  !horiz range from first xpndr
      R2 = RANGE(3)
C                                                  !horiz range from second xpndr
      DELTAX = TRX(3) - TRX(1)
      DELTAY = TRY(3) - TRY(1)
      DSQR = DELTAX*DELTAX + DELTAY*DELTAY
      TRDST = SQRT(DSQR)
C                                                  !baseline length
      R1SQR = R1*R1
      A = (R1SQR-R2*R2+DSQR) / (2.0 * TRDST)
C                                                  !projection of r1 onto baseline
      CSQR = R1SQR-A*A
C                                                  !fix to baseline dist. squared
      IF(CSQR .LT. 0) THEN
C                                                  !if neg. non-intersecting arcs
              ERR = 700.
C                                                  !set non-intersection code
              PRINT *,' Nonintersecting arcs for this fix.'
C                                                  !type nasty message
              CSQR = 0.0
C                                                  !place fix on baseline,
              RSUM = R1+R2
C                                                  !midway between the arcs.
              IF(RSUM.LE.TRDST) A=R1+(TRDST-RSUM)/2.0
              IF(RSUM.GT.TRDST) A=(TRDST+SIGN(RSUM,A))/2.0
      ENDIF
  100 CONTINUE
      C = SQRT(CSQR)
C                                                  !fix to baseline distance
      DX = DELTAX/TRDST
C                                   !cos of angle between baseline and horizontal
      DY = DELTAY/TRDST
C                                   !sin of angle between baseline and horizontal
```

```
Dec 21 19:51 1988  fix2.f Page 2

      XPRJ = TRX(ITR1) + A*DX
C                             !x-coord of projection of fix onto baseline
      YPRJ = TRY(ITR1) + A*DY
C                             !y-coord of projection of fix onto baseline
      CDX = C*DX
C                             !x displacement from (XPRJ,YPRJ) to fix.
      CDY = C*DY
C                             !y displacement from (XPRJ,YPRJ) to fix.
      X(1) = XPRJ - CDY
C                             !find the coordinates of the 2 intersections
      Y(1) = YPRJ + CDX
      X(2) = XPRJ + CDY
      Y(2) = YPRJ - CDX
      RETURN
      END
```

```
Dec 21 19:51 1988  xcor.f Page 1

      SUBROUTINE xcor ( xold, xdepth, vold, vdp, nvdps, xnew, z )
c     XCOR corrects a distance (xold) for an incorrect velocity (vold) and
c  applies a new one (vdp).
c     A travel time is calculated by assuming xold was computed with a
c  constant velocity (vold).  The new distance is calculated using a
c  summation of interval velocities.  The structure of VDP must be such
c  that a constant velocity is used for the interval between the depths on
c  either side in the vdp array.  This approach assumes that an XBT was used to
c  measure the instantaneous velocity for each depth and that the velocity
c  is constant for the interval between depth samples.  The velocity used
c  is the average velocity, assuming that the interval velocity is the
c  average between the 2 instantaneous velocities.
c      We know the ray is not vertical.  The initial angle can be measured
c  by using the known depth (z) and the slant range (xold).  If we assume
c  that the angle stays the same, we can figure out the length of time the
c  ray spends in each layer.
c      Snell's law or ray bending is ignored.
c      There's another hook in this problem!  That is, the distance that we are
c  correcting might not start at the surface.  The vdp array assumes that there
c  is a constant velocity from the surface to the first vdp depth.  XOLD might
c  have started at some depth because the device was sunk.
c
c  ARGUMENTS:
c  xold    - The distance to be modified.
c  xdepth  - The depth of the object where xold starts (the starting depth
c            of the ray).
c  vold    - The constant velocity used in obtaining xold.
c  vdp     - An array of velocity-depth pairs to use in calculating XNEW
c  nvdps   - The number of elements in the vdp array.  There must be
c            nvdps/2 pairs in the vdp array.
c  xnew    - The new distance calculated.  XNEW is returned by XCOR.  XNEW may
c            be the same as XOLD.
c  z       - The depth of the final point of the ray.
c
c          .... x ....
c          .         .xdepth
c          .        .
c          .       .
c          z     xold
c          .     .
c          .    .
c          .   .
c          .  .
c          . .
c          ..
c          .
c
c
      DIMENSION vdp(nvdps)
c
      theta = (z-xdepth)/xold
      IF( theta .LT. .0001 ) THEN
          rtheta = 1.
      ELSE
          rtheta = 1. / theta
      ENDIF
```

```
Dec 21 19:51 1988  xcor.f Page 2

      t = xold / vold                                                   ! find t
      x = 0.
      tleft = t
      IF( nvdps .EQ. 2 ) THEN                                           ! is the
          xnew = t * vdp(1)
          RETURN
      ENDIF
      IF( t*vdp(1) .LE. vdp(2) ) THEN                                   ! is it
          xnew = t * vdp(1)
          RETURN
      ENDIF
c
      idone = 0                                                         ! a flag
c****
c****  find the first vdp that is after xdepth
c****
      DO 50 k = 1, nvdps, 2
         i = k
         IF( xdepth.LT. vdp(k+1) ) THEN
             deltax = vdp(i+1)-xdepth                                   ! do the
             x = deltax * rtheta
             tleft = tleft - x/vdp(i)
             GOTO 190
      ENDIF
   50 CONTINUE
      xnew = t * vdp(nvdps-1)                                           ! must b
      RETURN
c****
  100 CONTINUE
      v = ( vdp(i) + vdp(i-2) ) / 2.                                    ! the in
      deltad = vdp(i+1) - vdp(i-1)                                      ! the le
      deltax = deltad * rtheta                                          ! the to
      deltat = deltax / v                                               ! the ti
      IF( tleft - deltat .GT. 0. ) THEN                                 ! does t
          x = x + deltax                                                ! add in
          tleft = tleft - deltat                                        ! subtra
      ELSE
          x = x + tleft * v                                             ! use th
          GOTO 200
      ENDIF
  190 i = i + 2
      IF( i .LT. nvdps ) GOTO 100
      x = x + tleft * vdp(i-2)                                          ! use th
  200 CONTINUE
      xnew = x
      RETURN
      END
```

```
Dec 21 14:57 1988  rdline.f Page 1

      SUBROUTINE RDLINE
C     RDLINE READS A LINE OF INPUT FROM THE STANDARD INPUT DEVICE (READER OR
C  TERMINAL.  THE LINE IS STORED IN LABELED COMMON Q$LINE AS A CHARACTER STRING.
C  THE STRING IS CLEARED TO BLANKS PRIOR TO READ.
C

      PARAMETER (MAXC=100)
      COMMON /SIOLN/ CBUF,ICHAR,NCHARS,iprint
      CHARACTER*100 CBUF


   10 CBUF(1:maxc)=' '
      READ (*,20) CBUF(1:MAXC)
   20 FORMAT(A100)
      nchars=0
      DO 30 i=1,maxc
         IF( cbuf(i:i) .NE. ' ') nchars=i
   30 CONTINUE
      cbuf(nchars+1:nchars+1)=' '
      cbuf(nchars+2:nchars+2)=char(0)
      IF( iprint .EQ. 1 ) PRINT *,cbuf(1:nchars)
      ICHAR=1
      RETURN
      END
```

Dec 21 14:57 1988 rline.f Page 1

```
      SUBROUTINE rline( lun )
C     RLINE READS A LINE OF INPUT FROM unit lun.
C  THE LINE IS STORED IN LABELED COMMON Q$LINE AS A CHARACTER STRING.
C  THE STRING IS CLEARED TO BLANKS PRIOR TO READ.
c
c  nchars is 0 if the end of file was detected or a blank line was read.
c
      PARAMETER (MAXC=100)
      COMMON /SIOLN/ CBUF,ICHAR,NCHARS,iprint
      CHARACTER*100 CBUF


   10 CBUF(1:maxc)=' '
      nchars = 0
      READ (lun, 20, END=100, ERR=100 ) CBUF(1:MAXC)
   20 FORMAT(A100)
      DO 30 i=1,maxc
         IF( cbuf(i:i) .NE. ' ') nchars=i
   30 CONTINUE
      cbuf(nchars+1:nchars+1)=' '
      cbuf(nchars+2:nchars+2)=char(0)
      IF( iprint .EQ. 1 ) PRINT *,cbuf(1:nchars)
      ichar = 1
  100 CONTINUE
      RETURN
      END
```

```
Dec 21 14:58 1988  upcase.f Page 1

      subroutine upcase(cbuf,n)        /* Vax Unix 4.2BSD Version
c       upcase converts n characters of cbuf to uppercase.  Upcase only converts
c lowercase alphabetic ASCII characters.
c
c  ARGUMENTS:
c    cbuf - The type character string to be converted.
c    n    - The number of characters in cbuf to convert. INTEGER*4
c
      character*1 cbuf(n)
      integer*4 n
c
      do 100 i=1,n
      if(ichar(cbuf(i)).ge.96.and.ichar(cbuf(i)).le.122)
     *     cbuf(i)=char(ichar(cbuf(i))-32)
  100 continue
      return
      end
```

Dec 21 15:01 1988   getoke.f Page 1

```
      SUBROUTINE GETOKE(CBUFO,NCHARS)
C
C    GETOKE RETURNS CONSECUTIVE TOKENS (ITEMS BETWEEN A DELIMITER), ONE PER CALL,
C    FROM A CHARACTER STRING (LINE OF INPUT). AN ALPHA STRING IS RETURNED IN
C    TOKEN WHEN IT STARTS AND ENDS WHEN SINGLE QUOTES.  (THE QUOTES ARE NOT
C    RETURNED.  THE STRING MUST BE TERMINATED WITH A QUOTE AND A BLANK, SO THAT
C    QUOTES MAY BE INCUDED IN THE STRING SO LONG AS THE QUOTE IS NOT FOLLOWED BY
C    A BLANK).
C
C   AGUMENTS:
C     CBUFO  - THE CHARCTER*1 ARRAY SET BY GETOKE CONTAING THE NEXT TOKEN IT
C              FOUND.  CBUFO MUST BE NCHARS+1 CHARACTERS LONG (SINCE C STRINGS
C              MUST BE TERMINATED WITH A NULL).
C     NCHARS - THE NUMBER OF CHARACTERS IN THE TOKEN RETURNED IN CBUFO. A 0
C              (ZERO) NUMBER OF CHARACTERS MEANS THAT NO TOKEN WAS FOUND AND
C              THAT ANOTHER LINE SHOULD BE READ AND GETOKE CALLED AGAIN WITH
C              ICHAR=1.
C
C

      CHARACTER*1 CDELIM,QUOTE,tab
      CHARACTER*100 CBUFIN
      COMMON /SIOLN/ CBUFIN,ICHAR,NCBUF
      CHARACTER* (*) CBUFO
      DATA CDELIM/' '/, QUOTE/''''/

      NCHARS=0                 /* COUNT THE NON BLANK CHARACTERS IN THE TOKEN
      IQUOTE=0                /* COUNT THE QUOTES IN THE TOKEN
      tab = CHAR(9)           /* the tab character
      IF(ICHAR.LT.1) ICHAR=1
   10 IF( CBUFIN(ICHAR:ICHAR) .NE. CDELIM .AND.
     *    cbufin(ichar:ichar) .NE. tab ) GOTO 20    /* STRIP OFF LEADING BLANKS
      ICHAR=ICHAR+1
      IF(ICHAR.GT.NCBUF) RETURN
      GO TO 10
   20 IF(CBUFIN(ICHAR:ICHAR).NE.QUOTE) GO TO 30           /* IS IT A QUOTE?
      ISTART=ICHAR+1                               /*STRIP OF THE LEADING QUOTE
      IQUOTE=1                      /* SIGNAL THAT THE STRING STARTED WITH A QUOTE
      GO TO 40
   30 ISTART=ICHAR            /* THE FIRST CHARACTER OF THE TOKEN TO BE RETURNED
   40 CONTINUE                              /* NOW FIND THE END OF THE TOKEN
      IF(ICHAR.GT.NCBUF) GO TO 100             /* ARE WE AT THE END OF THE LINE?
      IF(CBUFIN(ICHAR:ICHAR).EQ.CDELIM.AND.IQUOTE.NE.1) GO TO 100    /*  BLANK?
      ICHAR=ICHAR+1
      IF(CBUFIN(ICHAR:ICHAR).NE.QUOTE.OR.CBUFIN(ICHAR+1:ICHAR+1).NE.
     *    CDELIM.OR.IQUOTE.NE.1) GO TO 50   /* IS IT A QUOTE FOLLOWED BY A BLANK?
      IQUOTE=2
      GO TO 100
   50 NCHARS=NCHARS+1         /* THE CURREN CHARACTER IS NOT A BLANK OR A QUOTE
      GO TO 40                              /* GO LOOK AT THE NEXT CHARACTER
  100 CONTINUE
      IF(NCHARS.EQ.0) RETURN               /* DON'T TRY TO MOVE ZERO CHARACTERS!!
      IF(IQUOTE.ne.1) GO TO 110    /* CURRENT CHARCTER IS A BLANK WITHIN QUOTES
      ichar=ichar+1
      nchars=nchars+1
      go to 40
```

Dec 21 15:01 1988   getoke.f Page 2

```
c****  end the returned string with a null character so that c rcognizes
c**** the end of sting!!
  110 CBUFO(1:NCHARS)=CBUFIN(ISTART:ICHAR)
      cbufo(nchars+1:nchars+1)=char(0)
      ICHAR=ICHAR+1                                /* STRIP OFF THE BLANK
      RETURN
      END
```

Dec 21 15:04 1988 dcode.f Page 1

```
      SUBROUTINE DCODE(ALPHA,NCHAR,AREAL,ISTAT)
C     DCODE RETURNS A REAL NUMBER GIVEN A STRING OF CHARACTERS.  THIS RESEMBLES
C  THE OLD DECODE STATEMENT FOUND IN PRE-FORTRAN 77.  DCODE DOES NOT WORRY ABOUT
C  INTEGERS HOPING THAT THE CALLING ROUTINE CAN FIX IT IF IT NEEDS TO BE, AND
C  THAT ROUND OFFS WON'T HURT.  THIS ROUTINE ALSO HAS THE ADVANTAGE THAT IT DOES
C  NOT BOMB IF THE ALPHA IS NOT A NUMBER!  THE MESSAGE IS IN ENGLISH TOO!
C
C  ARGUMENTS:
C    ALPHA - THE STRING OF CHARACTERS TO BE CONVERTED TO AN INTERAL FLOATING
C            POINT NUMBER.  THIS MUST BE OF TYPE CHARCTER.   CHARACTER* (*)
C    NCHAR - THE NUMBER OF CHARCTERS IN THE STRING TO BE DECODED.  INTEGER*4
C    AREAL - THE FLOATING POINT (REAL) NUMBER DECODED BY DCODE.  THIS VALUE
C            IS RETURNED BY DCODE.   REAL*4
C    ISTAT - THE STATUS OF THE DECODE.    INTEGER*4
C          =0, THE DECODE HAD AN ERROR.  THE RETURN VALUE AREAL IS MEANINGLESS.
C              THE "NUMBER" HAD A NON-NUMERIC IN IT.
C          =1, THE CHARACTER STRING WAS AN ALPHA (THE FIRST CHARACTER WAS NOT
C              NUMERIC).  THE RETURNED VALUE OF AREAL IS MEANINGLESS.
C          =2, THE DECODE WAS SUCCESSFUL.
C
C
C
C
      CHARACTER*(*) ALPHA
      CHARACTER*20 CTEMP
C
      AREAL=0.
      ISTART=0
      JSTAT=0
      DO 100 I=1,NCHAR
      IF(ALPHA(I:I).GE.'0'.AND.ALPHA(I:I).LE.'9') GO TO 50     /* ASSUME ASCII!
      IF(ALPHA(I:I).EQ.'.') GO TO 100                    /* ALLOW A DECIMAL POINT
      IF(ALPHA(I:I).EQ.'-'.OR.ALPHA(I:I).EQ.'+') GO TO 100      /*SIGNED VALUES
      IF(ALPHA(I:I).EQ.'E'.OR.ALPHA(I:I).EQ.'e') GO TO 100      /* ALLOW EXPs

      ISTAT=1                                 /* PRESET TO AN ERROR

      GO TO 200
C
   50 ISTAT=2                                 /* THE CHARCTER IS A NUMERIC
      IF(I.EQ.1.OR.ISTAT.EQ.JSTAT.AND.ISTART.EQ.0) GO TO 90
      IF(JSTAT.NE.0) GO TO 200
   90 JSTAT=ISTAT      /* SAVE TYPE OF STRING TO COMPARE THE NEXT CHARCTER WITH
  100 CONTINUE
C
C    FINISHED SEARCH FOR ERRORS, NOW DECODE THE THING
C
      IF(ISTAT.EQ.1) RETURN                    /* DON'T DECODE AN ALPHA STRING
      CTEMP(1:20)=' '
      CTEMP(1:NCHAR)=ALPHA(1:NCHAR)
      IF( nchar .GT. 20 ) THEN
          PRINT *,' ***  WARNING  ***  The number ',alpha(1:nchar),
     *    ' exceeds the maximum field width of 20.'
      ENDIF
      READ(CTEMP,'(G20.0)',ERR=200) AREAL
      RETURN
C****
```

Dec 21 15:04 1988 dcode.f Page 2

```
C****   PRINT AN ERROR MESSAGE
C****
  200 PRINT 210, ALPHA(1:NCHAR)
  210 FORMAT('  ***  ERROR  ***  THE STRING ',A10,' IS NOT A NUMBER.')
      ISTAT=0
      RETURN
      END
```